MASTER IN COMPUTER SCIENCE

Co-analyse schéma-données-programmes en rétro-ingénierie des bases de données

Folisi, Piero

Award date:
2009

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année académique 2008 - 2009

Co-analyse Schéma-Données-Programmes en rétro-ingénierie des bases de données

Folisi Piero

Mémoire présenté en vue de l'obtention du grade de
Licencié en Informatique

Résumé

Le code procédural (requêtes SQL) d'une application est considéré depuis de nombreuses années comme une source d'information importante et précieuse pour la rétro-ingénierie de bases de données. En effet, les contraintes et structures de données, non déclarées explicitement lors de la conception de la base de données se retrouvent codées d'une manière ou d'une autre dans le code source des applications.

Un des objectifs de ce mémoire est de présenter une approche de rétro-ingénierie de base de données reposant sur les enchaînements d'opérations de manipulation de données relationnelles appelés « patterns ». Celle-ci tentera de rechercher et d'interpréter les patterns significatifs pour la recherche et la détection de contraintes implicites.

La rétro-ingénierie ne pouvant se limiter à l'exploitation d'une seule source d'information, le processus s'articulera autour de 3 sources d'information : l'analyse des patterns dans les programmes d'application; l'analyse du schéma physique et l'analyse des données.

Les contraintes potentielles dites « suspectes » obtenues à partir de l'analyse de ces différentes sources d'information devront être confrontées les unes aux autres afin d'atténuer certaines incohérences.

Le but de notre méthodologie est multiple; elle permettra avant tout d'expliciter les contraintes implicites, de détecter les données incohérentes, d'identifier les failles dans les programmes de manipulation de données et de proposer, si nécessaire, des améliorations tant du côté de la base de données que du côté applicatif.

Dans le but d'aider l'ingénieur dans les phases d'analyse du schéma et des données, des plugins d'assistance ont été développés à l'aide de l'API de DB-MAIN.

Mots-clés : Base de données, Rétro-ingénierie, Base de données relationnelle, Découverte de la sémantique de la base de données, Contraintes, Méthode, DB-MAIN, Schéma, Données, Patterns.

Abstract

For many years the source code (embedded SQL) of the application is one of the most accurate, relevant and up-to-date source of information for the database reverse engineering process. Indeed, many data structures and constraints that have not been explicitly declared during the conception phase, are coded, somehow or other in specific procedural sections of the application programs.

One goal of this document is to present a database reverse engineering approach to eliciting the semantic information stored in a relational database by analyzing sequences of data manipulation operations named "patterns". This one will try to recover and deduce implicit constraints by identifying some SQL/host language patterns that can be considered significant for the elicitation process.

To have homogeneous and viable results, the database reverse engineering cannot only explore one kind of information source. To capture a maximum of knowledge and semantics it is preferable to search for constraints using heuristics on other information source, therefore the process will take into consideration 3 relevant sources of information: query language statements (patterns) buried in application programs, data schema and data instances.

The result of the analysis of various sources of information, gives different assumptions concerning potential and suspicious constraints, to consolidate these assumptions, various indicators comings from the analysis process need to be confronted to each others to highlight some incoherence.

The aim of our methodology is rich and various, firstly to make explicit constraints implicit, secondly it will be used as a basis for detecting data inconsistencies and identifying unsafe program fragments and proposing necessary improvements at both the database and program sides.

In order to help the engineer during schema and data analysis, we have developed tools with help of the database reengineering environment DB-MAIN and his API.

Keywords: Database, Reverse engineering, Relational databases, Semantics discovery databases, Constraints, Method, DB-MAIN, Schema, Data, Patterns.

Remerciements

Un travail de fin d'études universitaires n'est pas une entreprise aisée.

Je tiens à exprimer ma reconnaissance et à remercier chaleureusement tous ceux sans qui ce travail n'aurait probablement pas pu être mené à bien.

En particulier Monsieur Jean-Luc Hainaut et Monsieur Anthony Cleve, respectivement promoteur et co-promoteur de ce mémoire, qui ont su me guider dans la rédaction de ce mémoire en m'accordant un temps précieux.
Leurs conseils m'ont toujours été d'un grand secours. Je les remercie également de m'avoir laissé une large part d'autonomie dans ce travail tout en m'aiguillant sur des pistes de réflexions riches et porteuses.

Un merci particulier à Madame Isabelle Linden. Vos remarques lors du séminaire préparatoire à la présentation des mémoires m'ont été fort utiles pour améliorer ce travail.

Je tiens à remercier également tous mes professeurs de la faculté informatique de Namur pour la qualité de l'enseignement qu'ils m'ont prodigué et qui m'ont préparé durant mes années d'études à aborder ce mémoire.

Je remercie l'ensemble de mes proches qui ont dû supporter mes humeurs et ma non disponibilité durant ces années d'études de cours du soir.
Sans eux, je n'aurais jamais réussi à atteindre l'étape du mémoire. Sans leurs sacrifices, je n'aurais jamais pu le terminer non plus. Je leur en serai éternellement reconnaissant et j'espère que l'avenir permettra de rattraper le temps perdu. Encore merci pour votre patience et votre dévouement.
Toute ma gratitude va à mes parents, qui n'ont jamais cessé de m'aimer et de m'encourager, je les remercie de m'avoir toujours donné une bonne éducation sans laquelle je n'aurais jamais pu arriver jusqu'ici.

Enfin, je souhaite remercier mes amis et toutes les personnes qui m'ont encouragé à aller jusqu'au bout.

Un grand merci à tous.

Table des matières

Table des figures

ANNEXES

Chapitre 1

INTRODUCTION

1.1 Introduction générale

Depuis les années quatre-vingt, un intérêt grandissant pour la rétro-ingénierie[1] des systèmes d'information a été observé et tout particulièrement dans le domaine des bases de données[2].

On trouve d'ailleurs de nombreux articles et ouvrages sur le sujet partant de fichiers classiques ([Casanova 1983], [Davis 1985], [Nilsson 1985], [Sabanis 1992], [Hainaut 1993b], [Edwards 1995]), de bases de données IMS[3] ([Navathe 1988], [Winans 1990], [Batini 1992], [Hainaut 1993b], [Fong 1994]), CODASYL[4] ([Batini 1992], [Hainaut 1993b], [Fong 1994], [Edwards 1995]), relationnelles[5] ([Casanova 1984], [Navathe 1988], [Davis 1988], [Johannesson 1990], [Markowitz 1990], [Spring 1990], [Batini 1992], [Fonkam 1992], [Chiang 1993], [Hainaut 1993b], [Shoval 1993], [Campbell 1994], [Chiang 1994], [Andersson 1994], [Petit 1994], [Premerlani 1994], [Signore 1994], [Vermeer 1995], [Comyn 1996], [Chiang 1996]) et orientées objet[6] ([Hainaut 1997a], [Theodoros 1998]).

L'intérêt de cette discipline peut trouver sa raison dans les coûts de plus en plus grandissants des activités auxquelles les entreprises doivent faire face tel que la maintenance de projet, l'évolution du système, la migration ou la conversion de données. Plus que jamais, les changements sont dans l'air du temps, plus que jamais les organisations doivent s'adapter et faire face à de nouveaux défis, comme l'était le bug de l'an 2000 ([Martin 1997], [IBM 1998]) ou le passage à l'Euro ([Gro 1998]).

---

[1] « La rétro-ingénierie (traduction littérale de l'anglais reverse engineering), également appelée rétro conception, ingénierie inversée ou ingénierie inverse, est l'activité qui consiste à étudier un objet pour en déterminer le fonctionnement interne ou sa méthode de fabrication » [Wikipedia].

[2] « Une base de données est un ensemble structuré de données enregistré sur des supports accessibles par l'ordinateur pour satisfaire simultanément plusieurs utilisateurs de façon sélective et en un temps opportun » [Delobel 1982].

[3] Information Management System, en français « Système de gestion de l'information », bases de données hiérarchiques créée par IBM en 1966.

[4] Conference on Data Systems Languages, en français « Conférence sur les langages de systèmes de traitement de données ».

[5] Base de données structurée suivant les principes de l'algèbre relationnelle [Codd 1970].

[6] Cas particulier des bases de données, dans une telle organisation, les données sont représentées sous forme d'objets [Wikipedia].

CHAPITRE 1 : INTRODUCTION

Le besoin d'évolution est et restera toujours présent; les entreprises devant s'adapter à de nouvelles lois, de nouvelles opportunités business, des fusions ou absorptions, des nouvelles technologies, des nouvelles architectures, etc.
Pour répondre à tous ces besoins grandissants de l'industrie, les applications de bases de données sont donc amenées à évoluer au cours de leur cycle de vie.

Mon expérience de plusieurs années dans le monde de la maintenance informatique au sein d'applications « mainframe » (écrites en langage COBOL[7] et utilisant des bases de données DB2[8]) m'a permis de me rendre compte que celles-ci étaient bien souvent pauvres en documentation, que dans bien des cas, cette documentation était perdue, n'était pas mise à jour ou n'avait même jamais existé.
De même, si une forme de documentation existait, celle-ci était bien souvent devenue périmée et obsolète en raison des nombreuses évolutions et modifications apportées au système.

Suite à ces manquements de documentation, nous sommes amenés à travailler directement sur les structures physiques qui manquent souvent de lisibilité. Du coup, le repérage des spécifications à modifier ou à implémenter dans le système prend plus de temps et demande des phases de tests plus importantes.

Chaque nouveau développement ou correction représente de ce fait un nouveau défi à relever ou énigmes à résoudre, lançant ainsi la quête d'informations vers la recherche d'indices, de traces et de connaissances à travers les composants de l'application, tels que l'expertise humaine, la documentation existante, les programmes, les données, le schéma, etc. Tous ces éléments pouvant aider à mettre à jour une partie de la sémantique enfuie et contribuer ainsi à la réussite du projet.

En effet, la bonne compréhension du système est un pré-requis indispensable avant de commencer toutes modifications. C'est d'ailleurs dans le besoin de pallier au manque et au non complétude de la documentation que la rétro-ingénierie tire son origine.

Toute cette recherche de connaissances et d'informations est complexe, mais importante pour la réalisation de nouveaux développements et pour la maintenance du système.

Dans ce contexte, on peut dire que la rétro-ingénierie des bases de données est vue comme un nouveau challenge pour la communauté des concepteurs. Elle devient donc un sujet de plus en plus attirant et intéressant dans le domaine de la recherche et de l'industrie. Même si Hainaut dans [Hainaut 2000] l'a définie comme le processus le moins excitant du domaine de l'ingénierie.

---

[7] Common Business Oriented Language. Langage de programmation né en 1959. Il est l'un des plus vieux langages encore utilisé. Son succès vient du fait qu'il est spécialement adapté à la résolution de problèmes de gestion commerciale.
[8] Système de gestion de base de données relationnelle d'IBM

L'étude et la démarche adoptée se limiteront au domaine de la rétro-ingénierie des bases de données relationnelles car elles sont le point central de beaucoup d'applications de gestion.

Dans l'état de l'art, nous présenterons brièvement différents auteurs dont les recherches et la méthodologie dans la rétro-ingénierie des bases de données ont contribué à la rédaction de ce mémoire.

1.2 L'approche

Notre approche présentera un aspect plutôt innovant, en respect avec les autres approches présentées dans la littérature, car basée sur l'interprétation et sur l'acquisition d'indices provenant de l'analyse du schéma (chapitre 4), de l'analyse des données (chapitre 5), et de l'analyse des patterns (chapitre 6).

Les heuristiques de l'analyse de schéma constituent la phase de découverte, elles permettent d'établir des hypothèses de départ.
Classiquement, ces hypothèses sont ensuite validées par l'analyse des données et par l'analyse des patterns. Nous n'excluons cependant pas le fait que l'analyse des patterns permet également de mettre à jour de nouvelles hypothèses.

Notre approche comportera également une phase dites de « qualité » (chapitre 8), offrant diverses solutions et idées pour garantir plus de robustesse à la base de données.

Notre étude sera orientée sur les fondements du modèle relationnel introduit par Codd en 1970 [Codd 1970]. Nous supposerons le lecteur familier avec les concepts généraux de la théorie de ce modèle.

Nos techniques d'analyse exploiteront les principales caractéristiques de ce modèle, modèle basé sur une structure de données simples et uniformes, à savoir la relation et bien ancré dans le monde des applications de bases de données.

En effet, cette technologie est la plus répandue et se retrouve dans pratiquement tous les domaines d'application. Elle demeure le moyen le plus populaire, le plus simple et le plus intuitif pour stocker, rechercher et manipuler des données.
De plus, son background théorique est plus large et plus enrichi que celui des bases de données réseaux et hiérarchiques (figure 1-1). Ce modèle est enseigné dans toutes les écoles, universités et formations professionnelles. Sa simplicité est due à une vision tabulaire des données.
Pour toutes ces raisons, il parait donc évident de consacrer notre approche à ce modèle.

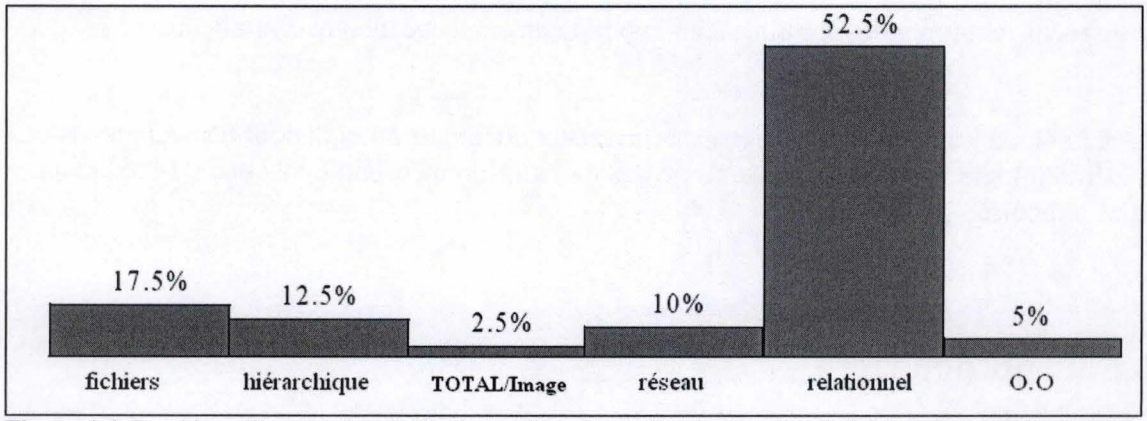


Figure 1-1 Graphique illustrant le modèle du système de gestion de données choisi lors de la publication en 2000 de 40 articles du monde des bases de données [Hainaut 2002].

1.3 L'objectif du mémoire

Le but principal de ce mémoire est de marquer par une petite empreinte le domaine de la rétro-ingénierie des bases de données en identifiant et en discutant des différents concepts de base relatifs à l'identification des contraintes implicites en explorant successivement différentes sources d'information .

L'analyse de cette variété de sources est importante. En effet, chacune d'elles est susceptible d'apporter des indices pertinents dans le processus d'élicitation.
La rétro-ingénierie ne pouvant se reposer sur l'exploration d'une seule source d'information. Que ce soit le schéma, les données, les spécifications LDD[9] ou les programmes, celles-ci peuvent être erronées ou incomplètes.

Ce travail présente donc une méthodologie, en développant un environnement d'assistance (plugins) relatif à la rétro-ingénierie des bases de données par l'intégration de trois composants majeurs : schéma physique, données et programmes.

Nous expliquerons pourquoi et comment nous avons choisi d'appliquer certaines techniques et outils dans le domaine de la rétro-ingénierie de bases de données et comment ceux-ci contribuent à la recherche d'indices aidant à la compréhension de la sémantique du système et de la base de données.

Bien que la rétro-ingénierie des données apparaisse comme une tâche complexe, la consultation de nombreux articles et ouvrages lors de la réalisation de l'état de l'art a contribué à enrichir notre bagage par de multiples concepts techniques et théoriques

[9] Langage de description des données. Un langage qui permet de décrire les données à intégrer dans une base de données.

permettant de rendre cette étude plus réaliste et plus accessible pour faire face à la résolution de problèmes.
Un chapitre contribuant à l'amélioration de la qualité sera également présenté.
Il montrera comment notre analyse des patterns, via son catalogue sera susceptible de corriger les modules sources d'éventuelles erreurs et comment l'analyse du schéma et des données permettra de mettre en place des mécanismes de contrôle et d'alerte lors de l'introduction de données ne respectant pas les règles du modèle, pouvant mettre en danger l'intégrité et la cohérence des données.

Accompagner et aider l'ingénieur dans ses tâches quotidiennes de développement, voilà l'objectif ambitieux de ce travail et la principale motivation qui nous ont poussés à choisir ce sujet de recherche.

1.4 Etat de l'art

1.4.1 Que s'est-il passé durant ces dernières années ...

Nous présenterons ici un résumé succinct de l'histoire de la rétro-ingénierie des bases de données.

Dans cet historique, nous avons introduit ce que nous estimions significatif dans la littérature de ces vingt-cinq dernières années.
L'objectif est ici de faire un rapide tour d'horizon concernant les recherches importantes ayant été accomplies dans le domaine de la rétro-ingénierie des bases de données.

Les articles et les auteurs cités ci-dessous, ne se veulent pas être une liste exhaustive, ils sont juste un exemple représentatif pour illustrer nos discussions sur le sujet.

Cet état de l'art a été réalisé en partie à l'aide du contenu historique se trouvant dans les documents de [Tari 1998], [Davis 2000], [Henrard 2003], [Lammari 2007] dont le résumé ce trouve ci-dessous.

Au départ, le domaine de recherche des projets de rétro-ingénierie des bases de données était basé uniquement sur la récupération des structures; les contraintes comme les identifiants et les clés étrangères étaient tout simplement ignorées.
Par la suite, les recherches ont porté sur la reconstruction de toutes les contraintes possibles.

Les premières méthodologies de rétro-ingénierie relatées dans la littérature étaient désireuses de produire un schéma conceptuel uniquement par l'analyse de la base de données. La principale source d'information disponible était donc le code LDD de la base, celui-ci était seulement raffiné par les connaissances de l'ingénieur.

Par la suite, beaucoup de recherches se sont focalisées sur différentes perspectives d'analyse pour comprendre la sémantique du système d'information. Celles-ci sont donc passées de l'analyse du schéma à l'analyse des requêtes ou des données.
En gros, on peut considérer qu'il existe trois générations de recherche.
La première avait pour objectif de proposer des démarches de rétro-ingénierie d'applications utilisant des fichiers conventionnels, de type COBOL ([Casanova 1983], [Nilsson 1985], [Davis 1988], [Hainaut 1991]).
La deuxième s'est intéressée plus principalement à la transformation des schémas de base de données navigationnelles de type réseau et hiérarchique ([Navathe 1988], [Winans 1990], [Batini 1992], [Hainaut 1993b], [Fong 1994]).

La troisième génération s'est attachée à la rétro-ingénierie des bases de données relationnelles en adoptant, soit une approche généralement formalisée de manière algorithmique ([Batini 1992], [Fonkam 1992], [Markowitz 1990]), soit une approche plus heuristique ([Hainaut 1992], [Chiang 1993], [Andersson 1994], [Petit 1994], [Premerlani 1994], [Signore 1994]).

Plus récemment, on constate un intérêt plus particulier de la rétro-ingénierie en vue d'une migration vers le monde objet ([Missaoui 1995], [Fong 1997], [Rama 1997]).

Dans cette section nous donnerons une brève vue d'ensemble de ces différentes approches en rapport avec notre méthodologie. Cet état de l'art du domaine a constitué un héritage précieux pour l'orientation de nos recherches.

Nos recherches étant focalisées sur le modèle relationnel, nous avons par conséquent orienté ce résumé historique uniquement sur les recherches concernant ce modèle.

1.4.2 Analyse du schéma relationnel

Beaucoup de recherches tombent dans cette catégorie d'approche et tentent de retrouver la sémantique par une analyse exhaustive de chaque relation et de leurs attributs contenus dans le schéma relationnel.

Dans cette catégorie d'analyse, nous retrouvons des auteurs comme Chiang ([Chiang 1993], [Chiang 1994], [Chiang 1995], [Chiang 1996]), Batini [Batini 1992], Markowitz [Markowitz 1990] et Johannesson [Johannesson 1990].

Leurs approches sont assez similaires; toutes partent de l'hypothèse que le schéma relationnel à analyser est en 3eme forme normale[10] facilitant ainsi la tâche du processus

[10] Une relation est en troisième forme normale si tout attribut contient une valeur atomique, si et seulement si il n'y a pas de dépendance fonctionnelle entre une partie de la clé et un constituant de la relation et que tout attribut n'appartenant pas à une clé ne dépend pas d'un autre attribut non clé [Hainaut 2006].

d'analyse. Le but final de ces méthodologies étant la conversion de schémas de bases de données relationnelles en schémas conceptuels (selon le modèle Entité/Association[11]).

Johannesson propose une approche basée sur la transformation du schéma relationnel (décomposition des relations, ajout des relations supplémentaires, fusion des relations et conversion en schéma conceptuel relationnel).

L'approche de Chiang est basée sur la classification des relations et des attributs, la génération et l'identification des dépendances d'inclusion.
Celle-ci utilise également les informations provenant des données pour parfaire son analyse.

Une heuristique souvent utilisée pour détecter la présence de relations entre les tables est de vérifier les similitudes entre le nom des colonnes, tout en vérifiant la compatibilité de leurs types et de leurs tailles.

De par sa simplicité de construction, le modèle relationnel ne peut exprimer certaines modélisations abstraites.

Cette sémantique perdue dans le schéma peut donc être retrouvée par la découverte d'autres indices éparpillés aux travers d'autres sources d'information comme les données ou les programmes.

1.4.3 Analyse des données

L'approche la plus connue est celle de Premerlani ([Premerlani 1994], [Blaha 1998]), basée sur les données et la connaissance de l'utilisateur.
Il s'agit d'un processus plutôt informel, suscitant beaucoup d'implication de la part de l'utilisateur.
Dans les étapes d'analyse, les clés candidates sont identifiées par la présence d'un index unique sur les attributs des clés, par la connaissance de la sémantique de l'utilisateur et par le scannage des données.

Les clés étrangères « potentielles », définies après un processus de « matching » sont déterminées également par cette analyse de données. Cette approche donne plusieurs indices sur la façon dont l'utilisateur devrait rechercher les types d'information requise.

1.4.4 Analyse des requêtes

Cette approche est basée sur la recherche d'indices pouvant détecter la présence de contraintes potentielles à partir des requêtes SQL se trouvant dans le code procédural.

[11] Modèle de représentation, il a été proposé pour la première fois par Chen [Chen 1976].

Les indicateurs obtenus en « parsant » les requêtes noyées dans le code de l'application détaillent la manière dont les données sont accédées et manipulées au sein du système d'information. Ils révèlent donc une bonne partie de la sémantique enfuie.

Dans cette catégorie d'analyse, nous retrouvons des auteurs comme Andersson [Andersson 1994], Petit ([Petit 1994], [Petit 1996a]) et Signore [Signore 1994].

Andersson reconstitue le schéma conceptuel par ses recherches sur l'utilisation des jointures contenant l'opérateur d'égalité (« equi-joins »[12]).

Petit, en plus de l'étude sur les « equi-joins », se focalise davantage sur les « auto-joins »[13] et sur les requêtes utilisant les clauses « where » et « group by » pour la détection des indicateurs sémantiques.

Le but principal est d'éliciter les clés étrangères et les dépendances fonctionnelles à partir des requêtes SQL embarquées.

Cette démarche, plus communément appelée analyse de patterns, sera abordée plus loin dans ce mémoire (chapitre 6).

1.4.5 Méli-mélo

Bien que les trois approches présentées ci-dessus englobent la plupart des processus d'analyse, il existe encore d'autres démarches importantes présentant des solutions pour le processus de rétro-ingénierie des bases de données.

Les recherches et les travaux faits par Hainaut ([Hainaut 1991], [Hainaut 1992], [Hainaut 1993a], [Hainaut 1993b], [Hainaut 1997a], [Hainaut 2000], [Hainaut 2002]) sont particulièrement intéressants pour comprendre les besoins de cette discipline.
Celles-ci détaillent par ailleurs un processus standard de conception de bases de données bien utile pour comprendre le processus de rétro-ingénierie.

1.4.6 Tableau comparatif des méthodes de rétro-ingénierie

Dans l'historique, nous avons pu constater que chaque méthode avait ses propres caractéristiques (entrée, sortie, hypothèses spécifiques, etc.).

Voici un tableau (figure 1-2) illustrant et comparant la plupart des méthodes reprises dans l'historique. La comparaison des méthodes de rétro-ingénierie nous permet plus facilement de voir laquelle peut être utilisée en rapport aux informations disponibles.

[12] Jointure permettant de relier, avec une relation d'égalité, des tables qui ont au moins un attribut commun.
[13] Jointure d'une table avec elle-même, cette opération est utile lorsque l'on souhaite relier des attributs qui se trouvent à l'intérieur d'une même table.

Les informations produites dans ce tableau sont issues de l'article [Pedro 1999].

Cette comparaison est faite ainsi :

Entrée – Les types d'informations nécessaires pour appliquer la méthode
Suppositions – Les suppositions pour chaque méthode
Sortie – Le(s) résultat(s) de la méthode
Méthodologie – Les étapes majeures suivies
Contribution majeure – La contribution majeure de cette méthode
Schéma – Se base sur le schéma pour appliquer la méthode
Donnée – Se base sur les données pour appliquer la méthode
Code – Se base sur le code applicatif pour appliquer la méthode
Problèmes – Problèmes constatés sur la méthode

)bjets/méthodes	**Chiang** [Chiang 1993], [Chiang 1994], [Chiang 1995]	**Johannesson** [Johannesson1994]	**Markowitz** [Markowitz 1990]	**Batini** [Batini 1992]	**Petit** [Petit 1994], [Petit 1996]	**Premerlani** [Premerlani 1994], [Blaha 1998]	**Signore** [Signore 1994]
ntrée	Instances de données, relations, clés primaires (Schéma phys.), données, connaissances	Relations, dépendances fonctionnelles, dépendances d'inclusion	Relations, dépendance de clés et des dépendances d'inclusion	Relations	Instances de données, relations, code	Relations (LDD[14]), données, connaissances	Relations(LDD), code
upposition	3FN[15], aucune erreur dans les données	3FN avec dépendances.	Forme Normale Boyce-Codd (FNBC-BCNF en anglais)[16]	Plusieurs hypothèses : 3FN ou FNBC, concordance sur les noms des attributs, toutes les clés candidates.	Aucune	Aucune	Aucune
ortie	MEA[17] étendu (EER)	Schéma conceptuel (Sans constructions objets complexes mais présence de généralisation)	MEA étendu (EER)	MEA étendu (EER)	MEA étendu (EER)	OMT[18]	MEA (E/R)

[14] Langage de description des données.
[15] Troisième forme normale.
[16] Une relation est en FNBC lorsqu'elle est en 3NF et que tous les attributs non-clé ne sont pas source de dépendance fonctionnelle vers une partie de la clé [Wikipedia].
[17] Modèle Entité Association: une représentation graphique de données composée de trois concepts: les entités (un objet concret ou abstrait de l'univers réel à représenter), les associations (regroupement d'entités dans lequel chaque entité joue un rôle précis) et les propriétés (les caractéristiques d'une entité) [Teo 1990].
[18] Object Modeling Technique (Modélisation et Conception Orientée Objet).

Méthodologie	Etapes : (1)classification des relations et des attributs, (2)génération et identification des dépendances d'inclusion, (3)identification des composants de MEA étendu	Etapes : (1)décomposition des relations, (2)ajout des relations supplémentaires, (3)fusion des relations, (4)conversion MCD[19] en schéma conceptuel relationnel	Etapes : (1)transformation des MCD, (2)contrôle des relations et des clés, (3)détermination du type d'interaction, (4)construction d'un schéma MEA candidat	Classification et traitement des associations par intervention de l'utilisateur, mapping des associations, manipulation des cas spéciaux	Proposition des dépendances d'inclusion, proposition des dépendances fonctionnelles appropriées, obtention de la 3FN de la 1FN	Préparation du modèle objet, recherche des clés candidat, détermination des clés étrangères, amélioration du schéma OMT	Etapes : (1)identification des clés primaires, (2)détection des indicateurs des synonymes, (3)construction du schéma conceptuel (conceptualisation)
Contribution	Génération des dépendances d'inclusion, production du MEA étendu	Le plus complet mais a besoin de toutes les clés et dépendance d'inclusion.	Indépendance des noms des attributs, transformation des règles de mapping entre les schémas.	Résout la plupart des situations courantes	La seule méthode qui inclut la normalisation du schéma de la relation	Donne plusieurs indices sur la manière dont l'utilisateur devrait rechercher les types d'information requis	Adaptable pour les techniques d'exécutions rares, où on a un LDD faible ou des erreurs de code.
Schéma	Oui	Oui	Oui	Oui	Non	Non	Oui
Donnée	Oui	Non	Non	Non	Oui	Oui	Oui
Code	Non	Non			Oui	Non	Oui
Problème(s)	Ne peut être appliquée que si les données de la base sont la plupart du temps correctes et représentatives	Emploie seulement l'information du schéma, présupposant une grande connaissance de la structure et du contenu à partir des sources disponibles			Ne peut être appliquée que si le code et les données sont représentatifs	Basée sur les données et la connaissance de l'utilisateur	Ne peut être appliquée que si le code est représentatif et les utilisateurs ont une connaissance suffisante du domaine d'application

Figure 1-2 Tableau de comparaison des méthodes de rétro-ingénierie

[19] Modèle Conceptuel de données.

1.5 Notre approche

Comme montré ci-dessus, il est relativement simple d'extraire une partie de l'information en partant d'une seule source. Par contre, obtenir toute l'information en partant d'une seule source est quasi impossible.

A la différence de ces approches, la démarche présentée dans ce mémoire utilise et combine de manière sélective et intensive différentes sources d'information.

Cette imbrication des trois catégories d'approche s'apparente un peu à la méthode présentée dans les travaux de Signore ([Signore 1994], [Signore 1994b]). Celui-ci combine différentes sources d'information et utilise également la notion de patterns.

La pratique et les articles émanant de la littérature scientifique montrent que nous devons combiner un ensemble de règles, de méthodes, d'outils et de techniques de rétro-ingénierie afin d'être à même d'extraire et de confronter une bonne partie de la sémantique cachée du système.

De plus, choisir une méthode qui convienne au processus de rétro-ingénierie de bases de données est une tâche non triviale. Celle-ci a souvent différents pré-requis en entrée et chaque application a ses propres caractéristiques.

Le département de recherche de bases de données de Namur propose également une méthodologie d'approche intéressante dans ses différents articles du domaine de la rétro-ingénierie des bases de données [Hainaut 1993a]. C'est d'ailleurs cette méthodologie qui nous a orienté et guidé dans cette étude.

Notre approche sera détaillée dans le chapitre 3 de ce mémoire.

1.6 Grandes lignes du mémoire

Dans un premier temps, nous allons faire un rapide tour d'horizon et présenter les concepts de la rétro-ingénierie des bases de données (chapitre 2) pour ensuite définir de manière précise notre méthodologie d'approche sur l'élicitation des contraintes implicites à partir de différentes sources d'information (chapitre 3).
Nous présenterons et illustrerons à l'aide d'exemples les principales méthodes d'analyse utilisées et les plugins développés dans le cadre de notre approche (chapitres 4 à 6).
Afin de montrer la complémentarité de ces processus, nous décrirons comment il est possible grâce aux différentes hypothèses issues des phases d'analyse d'éliciter une clé étrangère (chapitre 7).
Le chapitre 8 tentera de rendre l'application plus robuste aux modifications des données, en proposant diverses solutions tant du côté applicatif que du côté de la base de données.
Pour finir, le chapitre 9 sera dédié à la conclusion et aux perspectives futures.

En annexe nous proposerons un descriptif et une illustration de l'implémentation de nos plugins mis en œuvre dans le cadre de ce travail pour faciliter la tâche de l'ingénieur dans le processus de rétro-ingénierie.

1.7 Limitation

Avant d'introduire notre démarche, il nous semblait utile de spécifier certaines limitations importantes de notre méthodologie, la rétro-ingénierie est une tâche vaste et complexe qu'il serait illusoire d'étudier dans sa globalité.

Hypothèses restrictives et limitations :

- Nos recherches de constructions implicites se focaliseront sur un nombre limité d'objectifs centrés sur la découverte des contraintes d'intégrité référentielle (clé étrangère[20]) et des identifiants (clé primaire).
- Notre analyseur de schéma, considérera uniquement les schémas en 3FN.
- Notre analyse de patterns, se fera de façon manuelle et statique, aucun outil de recherche ne sera développé dans ce mémoire.
- Notre approche aura besoin de gagner en expérience car elle n'aura pu être testée que sur peu d'applications concrètes.

Dans la rédaction de ce mémoire, nous utiliserons les concepts du modèle relationnel [Codd 1970] et de la terminologie du modèle Entité/Association [Teo 1990].

[20] Nous ne tiendrons pas compte des auto-jointures dans notre analyse.

Chapitre 2

RETRO-INGENIERIE DES BASES DE DONNEES

Avant d'évoquer notre méthodologie, nous devons d'abord apporter quelques précisions sur certains concepts importants de la rétro-ingénierie.

Depuis le bug de l'an 2000, l'Euro et l'introduction des nouvelles technologies, l'intérêt pour la rétro-ingénierie n'a cessé de grandir. En effet, ces grands changements ont suscité beaucoup de questions. Par conséquent, l'intérêt des informations révélant une partie de la sémantique de l'application n'a cessé de prendre de la valeur.

Après vingt-cinq ans d'histoire, nous devons regarder la rétro-ingénierie comme une partie essentielle de l'ingénierie logicielle et admettre qu'elle est devenue plus qu'une nécessité.

Rentrons un peu plus dans les détails de cette discipline.

2.1 Introduction et définition

Selon [Hainaut 2000] et [Henrard 2002] « *La rétro-ingénierie d'un composant logiciel est un processus d'analyse visant à reconstruire les spécifications fonctionnelles et techniques à partir de la version opérationnelle de ce composant* ».

Pour [Chikofsky 1990] la rétro-ingénierie est vue comme « *un processus se concentrant sur les données du système de l'organisation. C'est une collection de méthodes et d'outils visant à aider une organisation dans la détermination de ses structures et fonctions ainsi que dans la compréhension de ses données* ».

Ces deux définitions définissent bien le contexte de travail dans lequel nous nous trouvons.

Pour notre domaine dirigé vers les applications orientées données, c'est-à-dire les applications dont le composant central est une base de données, la rétro-ingénierie est vue comme « *un processus qui recouvre le schéma des données persistantes d'une application en partant de l'existant afin de comprendre la sémantique et la structure de l'application* » [Hainaut 1993a].

2.2 Objectif et utilité

La rétro-ingénierie a pour objectif de reconstruire les modèles de données. Pour atteindre cet objectif elle s'appuie sur un ensemble disponible de connaissances, tels que l'expertise humaine, la documentation existante et les éléments techniques.

Son utilité est multiple. On la retrouve dans des activités telles que la (re)documentation, l'administration des données, la maintenance, l'extension, la réutilisation et l'intégration d'applications, mais également dans l'évaluation de la qualité, la conversion, la migration et l'extraction des données.

D'ailleurs une liste exhaustive présentant les raisons de la pratique de la rétro-ingénierie a été présentée par Hainaut lors de son workshop à Zurich en Mars 2000 [Hainaut 2000].

La rétro-ingénierie vise à reconstituer le schéma de la base de données opérationnelle en retrouvant les constructions implicites (structure et contraintes) n'ayant pas été explicitement déclarées dans la base de données en utilisant différentes techniques d'analyse.

Son but premier est donc de récupérer les spécifications techniques et fonctionnelles de l'application existante et de créer, sous une autre forme, une nouvelle représentation du système ou de le représenter à un plus haut niveau d'abstraction (figure 2-1).

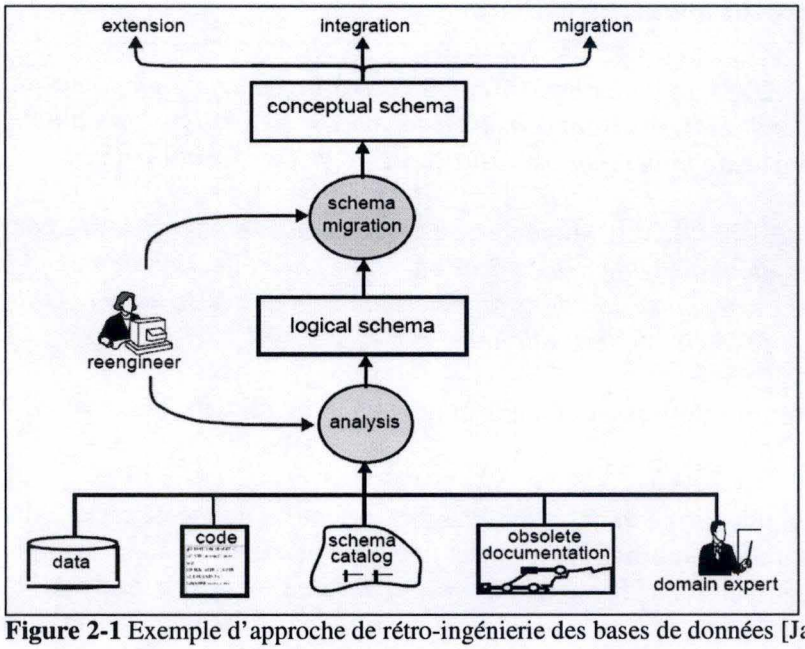


Figure 2-1 Exemple d'approche de rétro-ingénierie des bases de données [Jah 1999].

2.3 Complexité de la tâche

On pourrait penser que la rétro-ingénierie des bases de données est un processus moins complexe car il se focalise uniquement sur les données.

Mais depuis que le langage de description de données (LDD) n'est plus la seule source d'information, l'ingénieur doit analyser tout un tas d'autres composants du système. Le processus devient donc plus que complexe.

Toute une série de problèmes augmentant la complexité de la tâche de rétro-ingénierie ont été identifiés par [Hainaut 1993a], [Premerlani 1994], [Blaha 1995], [Anderson 1996], [Petit 1996b] comme la présence de structures et de contraintes implicites, de structures obsolètes, de la grande taille du système, du non respect des standards, de la présence de structures optimisées[21] ou d'implémentation maladroite due à des novices n'ayant pas assez de connaissances dans la théorie de base de données.

Le problème devient d'ailleurs d'autant plus long et complexe que l'application est ancienne, mal conçue, mal structurée ou pauvre en documentation.

Dans de nombreuses entreprises l'information est rarement structurée de façon normalisée et cohérente. Ceci est dû aux premiers systèmes de gestion de bases de données (SGBD)[22] qui n'étaient pas d'une grande souplesse.

Le manque de méthodes de développement (manque de guidelines, utilisation d'astuces techniques au détriment de la lisibilité) conduit souvent à un code illisible et obscur.

Ce processus est considéré depuis longtemps comme une activité pénible, longue et risquée. Il s'apparente à un parcours semé d'embuches en raison de ses aspects économiques, de ses ressources limitées, de son manque d'information fiable, de sa difficulté à évaluer le résultat produit, de l'existence de nombreux outils et techniques disponibles, etc.

Un autre problème est que la majorité des propositions de méthodes impose des hypothèses souvent trop idéalistes et restrictives tel que « le schéma n'a pas subi de restructuration d'optimisation », « les noms sont significatifs et rationnels», « le schéma est minimum en 3FN », « tous les besoins conceptuels ont été traduit dans le code LDD ou en contraintes», « un schéma physique complet est disponible », « présence des identifiants », etc.

Or, une part importante des applications violent ces hypothèses.

---

[21] Constructions non normalisées et redondantes ajoutées pour améliorer le temps de réponse.
[22] Système de gestion de base de données, c'est un ensemble de logiciels qui sert à la manipulation des bases de données [Wikipedia].

Dans le meilleur des cas une partie de ces contraintes a été documentée ou a été gérée via le code LDD, via les déclencheurs ou procédures du SGBD.
Cette situation idéale ne peut être certifiée dans tous les systèmes de bases de données, surtout dans les plus anciens, souvent pauvres en construction sémantique (absence de la notion de clé étrangère, de déclencheurs, de procédures, etc.). Il convenait donc à l'ingénieur de les implémenter dans le code applicatif.

Rappelons également que le modèle relationnel ne supporte pas tous les constructeurs du modèle conceptuel, dès lors une partie de la sémantique du schéma conceptuel est nécessairement perdue lors du passage au schéma relationnel.

Toute cette série de problèmes non triviaux, généralement rencontré sur le terrain, nous permet d'affirmer que ce processus n'est pas si simple, sauf peut-être dans certaines situations accidentellement favorables ou très simplistes.

La tâche, bien que complexe, n'est pas insurmontable. Actuellement la rétro-ingénierie est un processus bien maîtrisé et de nombreuses techniques sont mises à notre disposition pour faire face à cette complexité. De plus, de nombreuses sources d'information peuvent contribuer à l'élicitation de dépendances et sémantiques cachées.

C'est pourquoi, l'utilisation de techniques et d'outils adéquats va rendre l'étude moins complexe, moins longue et moins coûteuse.
Bien que la rétro-ingénierie ne soit pas totalement automatisable, les outils, processus et méthodes ont un grand potentiel et permettent d'automatiser ce qui l'est, apportant ainsi une aide précieuse à l'ingénieur pour retrouver l'information nécessaire et l'assister dans ces choix.

Cependant, affirmer pouvoir retrouver toutes les spécifications par les processus de rétro-ingénierie serait irréaliste.

2.4 Une méthode générique de rétro-ingénierie

Pour nous aider, une méthode générique de rétro-ingénierie de bases de données existe (figure 2-2) et nous servira de base pour mieux comprendre ce domaine complexe.
Précédée d'une phase de préparation, elle comporte deux processus majeurs basés sur le processus d'analyse de conception des bases de données.

Ses deux processus sont :

- L'extraction des structures de données (Data structure extraction)
- La conceptualisation des structures de données (Data structure conceptualisation)

Une analyse plus approfondie de la démarche est présentée dans [Hainaut 1993a] et [Hainaut 1997b].

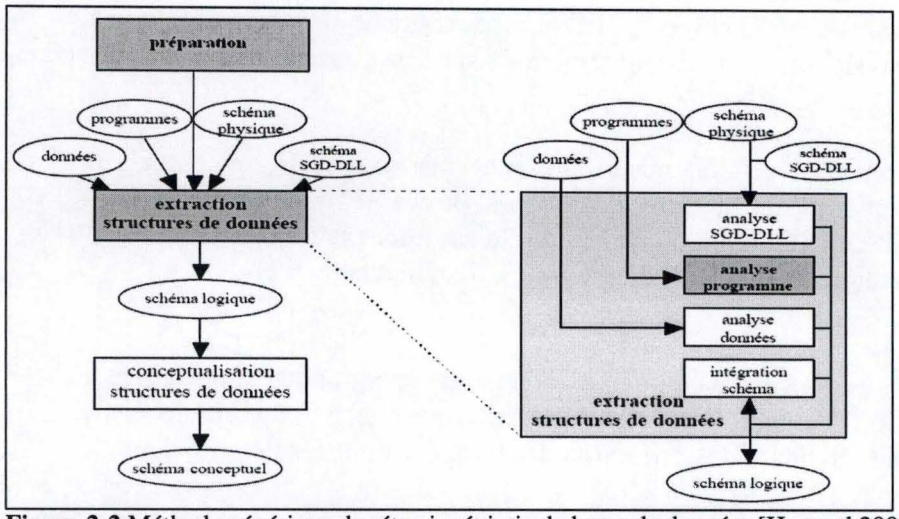


Figure 2-2 Méthode générique de rétro-ingénierie de bases de données [Henrard 2002].

Le processus de rétro-ingénierie de base de données est en fait l'inverse du processus de conception de base de données (figure 2-3). De façon similaire à tout processus d'ingénierie de base de données, elle doit prendre en compte un ensemble de modèles. Ces modèles doivent être capables de décrire des structures de données à des niveaux d'abstraction différents, du niveau physique au niveau conceptuel et selon des paradigmes de modélisation divers.

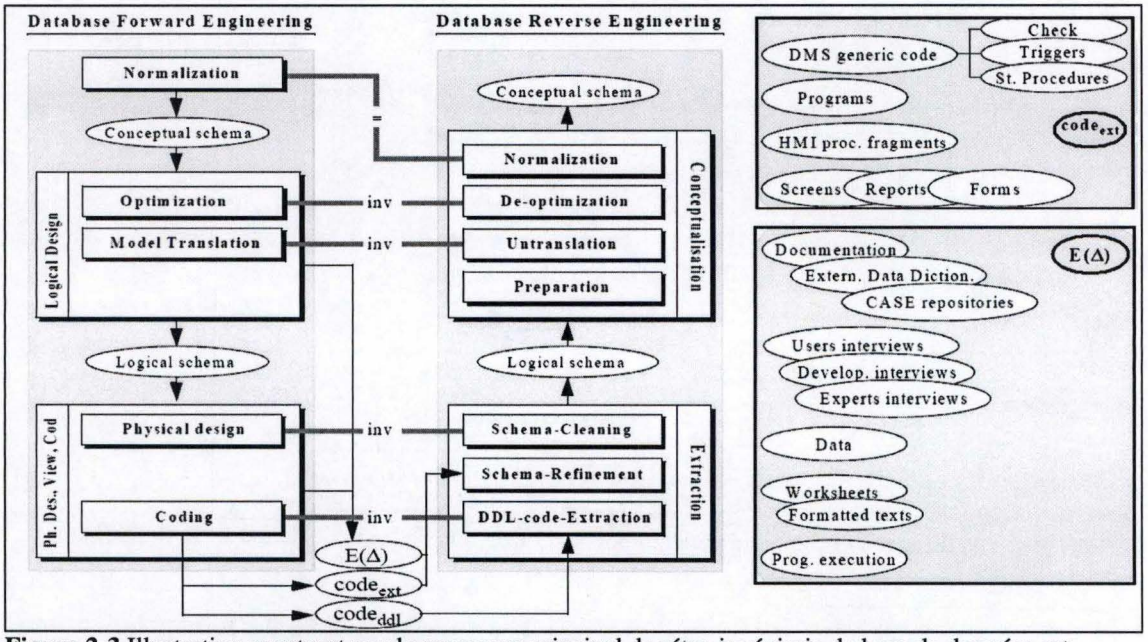


Figure 2-3 Illustration montrant que le processus principal de rétro-ingénierie de base de données est l'inverse du processus de conception [Hainaut 2002].

Grosso modo, on peut dire que le processus de rétro-ingénierie comprend trois étapes principales: (i) le recouvrement du schéma physique à partir du système opérationnel (les informations internes comme le code LDD et les informations externes comme les programmes, les triggers, les données, etc.); (ii) le recouvrement du schéma logique à partir de ce schéma physique; (iii) le recouvrement du schéma conceptuel à partir de ce schéma logique.

Les résultats finaux sont donc généralement représentés par un modèle conceptuel, toutefois, il peut également s'agir d'autres documents. En réalité, toute forme de documentation produite, facilitant ou pouvant aider la compréhension du système actuel est acceptable (dictionnaire de données, diagrammes, documents textuels, etc.).

La rétro-ingénierie est un processus complexe, il demande une grande connaissance du processus de conception[23] (figure 2-4). Nous encourageons le lecteur à s'intéresser de plus près à ce processus standard de conception des bases de données détaillé dans [Hainaut 2002].

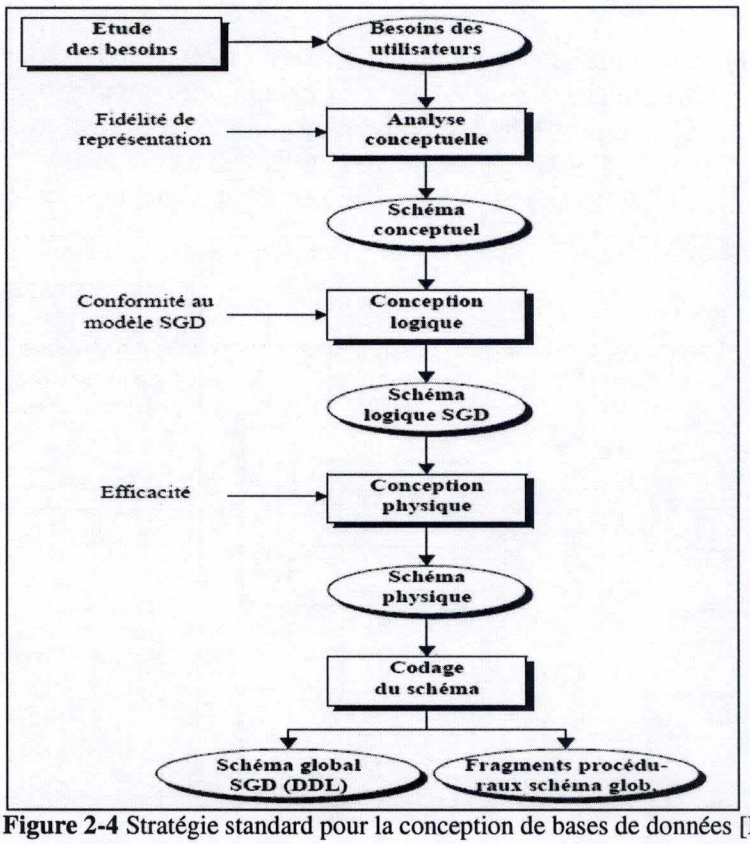


Figure 2-4 Stratégie standard pour la conception de bases de données [Hick 2001].

[23] La conception de bases de données est un processus qui transforme l'expression des besoins des utilisateurs en une collection de schémas et de programmes qui respectent des critères de correction, d'efficacité, d'opérationnalité, etc. [Hick 2001].

Le domaine d'approche de ce mémoire se situe au niveau du processus d'extraction des structures de données, qui a pour objet la reconstruction du schéma logique complet comprenant les contraintes et structures de données non explicitement déclarées en partant de diverses sources d'information (données, programmes, schéma physique, code LDD, etc.).

La phase de conceptualisation des structures des données est hors sujet.
Celle-ci interprète le schéma obtenu lors de la phase d'extraction des structures des données pour en dériver un schéma conceptuel. Elle détecte et transforme les redondances et les structures non conceptuelles introduites lors de la conception de la base de données [Henrard 2002]. Ce processus englobe la normalisation, la détraduction et la désoptimisation. Il transforme le schéma logique en schéma conceptuel.

2.5 Les outils

Il existe beaucoup de techniques, d'approches et d'outils supportant le processus d'extraction comme la recherche dans les textes sources (« text analyser »), la fragmentation de programme (« program slicing »), le graphe de dépendance (« dependency graph »), ainsi que de nombreux assistants de recherche de clés étrangères et de recherche de patterns (« pattern matching »).
Quelques uns de ces outils sont décrit brièvement dans [Hainaut 2002] et dans [Henrard 2003].

Nous nous focaliserons sur les techniques les plus populaires et touchant à différentes sources d'information (schéma, données, programmes).

2.6 Ce qu'il reste à faire ...

Dans le domaine de la rétro-ingénierie, beaucoup de problèmes ont déjà été identifiés. En effet, cette discipline dispose de techniques performantes pour accomplir ces tâches. Cependant, pour mener à bien des projets de grandeur réelle, ce processus réclame des outils puissants, flexibles et adaptés car les sources d'information sont volumineuses et diversifiées [Hainaut 2000].

Pour les perspectives futures de cette discipline, certains points importants ont été abordés par Hainaut dans [Hainaut 2000], mentionnant le chemin qui restait encore à parcourir comme l'apprentissage, le développement de méthodes et d'outils évolutifs, le raffinement des heuristiques, etc.

N'oublions pas que les coûts liés au processus de rétro-ingénierie sont un point critique en particulier dans le monde de l'entreprise. Il est difficile aujourd'hui d'évaluer son coût car nous ne savons pas quand l'arrêter.
Le processus prend-il fin parce que nous disposons d'assez d'informations ou parce que les budgets ont été dépassés ?
Beaucoup d'outils créés dans le cadre de la rétro-ingénierie bien qu'utiles semblent extrêmement limités.
L'atelier DB-MAIN[24] présenté brièvement en annexe essaie de couvrir et d'aider au mieux ce processus. Il est l'un de ceux qui continue à évoluer et donc à être utilisé.

2.7 Conclusion

L'importance de la compréhension du processus de rétro-ingénierie et de ses différentes approches a toujours été notifiée par de nombreux chercheurs.
Le parcours littéraire de ces différents articles et ouvrages a permis de tirer de nombreuses leçons et informations sur le sujet et de surcroît une méthode d'approche a pu être élaborée dans le chapitre suivant.

[24] DB-MAIN est un outil dédié à l'ingénierie des applications de bases de données [DBMAIN].

Chapitre 3

METHODOLOGIE DE L'APPROCHE

3.1 Les contraintes

3.1.1 Contraintes Implicites/Explicites

Le problème principal dans le domaine de la rétro-ingénierie concerne les contraintes dites « implicites », c'est-à-dire celles qui n'ont pas été traduites explicitement sous la forme de code déclaratif et qu'on retrouve à travers les données. Elles sont généralement cachées dans les programmes, les écrans, les formulaires, etc.
Contrairement aux contraintes dites « explicites » qui, elles se retrouvent dans le code déclaratif de la base de données.

Il serait idéaliste de penser que toutes les contraintes se retrouvent codées de manière explicite, c'est peut-être le cas pour certaines bases de données académiques, mais dans le monde réel, c'est loin d'être le cas.

Les constructions explicites peuvent être facilement et automatiquement analysées.
Elles sont déclarées dans le langage de description de données (schéma 3-1).
L'analyse de ce code est d'ailleurs étudiée depuis les années 80 et il existe beaucoup d'outils pour réaliser cette tâche.
Les constructions implicites sont plus difficilement identifiables car implémentées (ou non) dans d'autres parties de l'application, celles-ci doivent donc être découvertes.

Cette étude se concentrera principalement sur la recherche des contraintes implicites.

Si nous considérons le code LDD de la figure 3-1, on observe clairement la déclaration de deux tables avec une clé étrangère; ceci est donc clairement une contrainte explicite.
Par contre, si nous prenons le code de la figure 3-2, il montre la définition des deux mêmes tables que dans l'exemple précédent mais sans déclaration de clés étrangères.
Celles-ci sont définies dans le code applicatif; c'est ce qu'on appelle une contrainte implicite.

C'est là que réside toute la difficulté de la rétro-ingénierie. Celle-ci doit être capable de trouver des indices de poids, indices qui se rejoignent afin de constituer une information consistante afin de rendre plus compréhensible la sémantique des données.

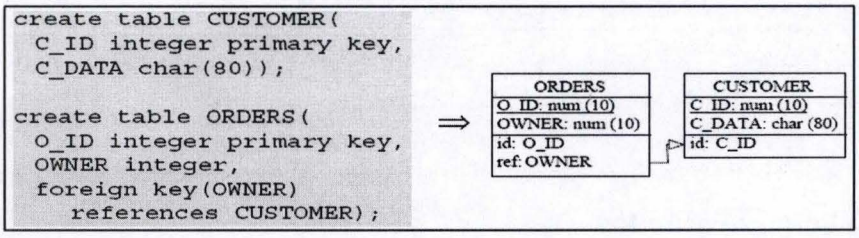


Figure 3-1 Déclaration de tables avec une clé étrangère explicitement définie dans le code SQL-LDD [Henrard 2002].

```
create table CUSTOMER(
 C-ID integer primary key,
 C-DATA char(80))
create table ORDER(
 O-ID integer primary key,
 OWNER integer)
```

```
exec SQL
 select count(*) in :ERR-NBR from ORDER
  where OWNER not in
   (select C-ID from CUSTOMER)
end SQL
...
if ERR-NBR > 0 then
 display ERR-NBR,
   'referential constraint violation';
```

a) Déclaration de tables sans clé étrangère. b) Fragment de code vérifiant la validité de la clé étrangère.

Figure 3-2 Déclaration de tables avec une clé étrangère implicitement définie dans le code applicatif [Henrard 2002].

Nous tenterons d'éliciter, c'est-à-dire de rendre explicite ces contraintes implicites par une analyse méticuleuse des sources d'information disponibles.
En effet, ces contraintes ont dû être exprimées sous une forme procédurale, graphique ou physique ou elles ont peut-être tout simplement été ignorées ou perdues.

La recherche de ces contraintes cachées est donc vitale pour permettre d'enrichir la documentation afin de la rendre plus précise et complète et de constituer ainsi une base solide pour la maintenance et la cohérence future de la base de données.

Pour les détecter, nous utiliserons tous les indices que nous pourrons extraire par notre méthodologie et par nos techniques d'analyse. Il va de soi que, pour la recherche d'indices, il nous faut puiser l'information tant du côté applicatif que du côté de la base de données. Tous ces indicateurs nous mènerons à la découverte des contraintes implicites.

3.1.2 Origine des contraintes implicites

Leur apparition est le résultat de plusieurs facteurs.

Bien souvent, les anciens systèmes de gestion de base de données étaient peu expressifs, n'offrant pas la possibilité de définir certaines contraintes, il appartenait donc à l'ingénieur de les gérer et de les implémenter d'une autre manière.
Celles-ci sont alors codées de manière décentralisée et se retrouvent gérées le plus souvent dans les programmes.

D'autres systèmes de gestion de bases de données, bien que plus récents et donc moins pauvres en expressions sémantiques, peuvent présenter également peu ou pas de contraintes explicites et ce, pour divers motifs; raisons d'efficacité, de portabilité, de sécurité, de facilité ou bien parce que l'ingénieur a décidé de les implémenter dans le code procédural car celui-ci était moins expérimenté ou familiarisé avec la théorie des bases de données.

Certains systèmes de gestion de base de données offrent la possibilité de gérer les contraintes via les vues (« with check option »), via les mécanismes de triggers ou de procédures ou en utilisant les prédicats « table check ». Ces techniques permettent de centraliser la gestion des contraintes mais peuvent être codées de manière fort différentes et les éliciter devient donc une tâche complexe.

3.1.3 A la recherche des contraintes implicites ...

Quelles sont donc les pistes à envisager pour les retrouver ?

Mis à part certains petits projets, généralement plus d'une source d'information doit être analysée pour trouver les propriétés cachées de la base.

L'ingénieur ne peut donc pas se limiter à une seule source d'information, il doit parcourir tous les composants possibles entourant l'application.

Voici une liste regroupant les sources d'information les plus courantes que l'on peut utiliser pour retrouver la trace de contraintes implicites (figure 3-4) :

> *Trigger/procédure* : Mécanismes internes de la base de données permettant la gestion de contraintes à l'intérieur du SGBD.
>
> *Programme (statique)* : Beaucoup de contraintes et structures non déclarées sont codées dans les programmes (figure 3-2). C'est l'une des sources d'information la plus importante.

Programme (dynamique) : La recherche de contraintes se fait par une analyse dynamique des codes sources des programmes (log, traces), on étudie leurs exécutions (« run-time ») [Cleve 2008].

Données : Les données permettent de restituer des indices qui peuvent être utilisés pour valider ou réfuter des hypothèses émises sur des contraintes.

Schéma physique : Résultats de l'analyse du code LDD, le schéma est une source d'information comprenant généralement les structures et contraintes explicites.

Ecran, rapport, formulaire : Ceux-ci peuvent être considérés comme une vue des données. Leurs présentations et leurs labels révèlent des informations essentielles sur les données (figure 3-3).

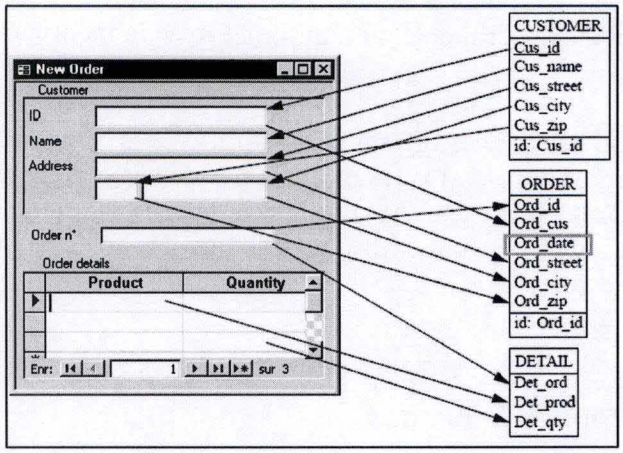


Figure 3-3 Exemple d'une source d'information : écran [Henrard 2002].

Documentation : Bien que la documentation puisse être partielle, obsolète et souvent incorrecte, elle contribue à donner des informations précieuses. Le code source commenté est également une riche source de documentation.

Connaissance du domaine : Il est inconcevable de commencer un projet de rétro-ingénierie sans aucune connaissance de l'application. En effet les connaissances de l'ingénieur lui permettent de valider ou de réfuter les indices issus des différentes phases d'analyse lors de la recherche de contraintes.

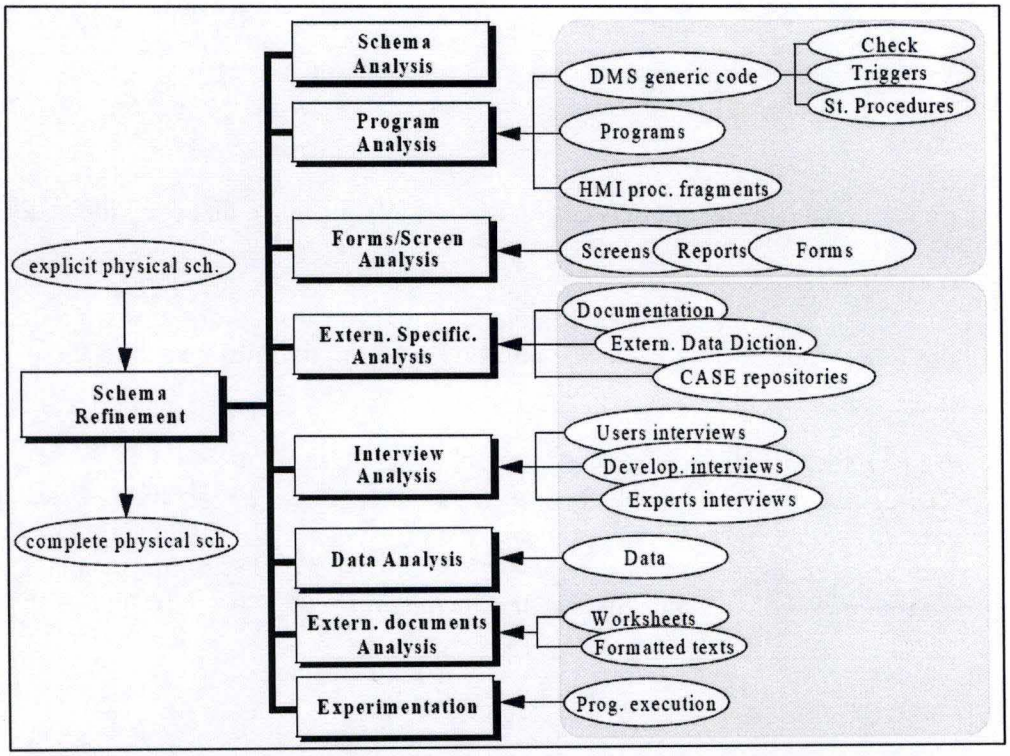


Figure 3-4 Enrichissement du schéma physique à l'aide de différentes sources d'information [Hainaut 2002].

3.1.4 Les différents types de contraintes

Une contrainte n'est autre qu'une règle impérative ne devant en aucun cas être violée. Elle assure l'intégrité et la cohérence des données.

Dans la littérature relatant de la théorie des bases de données, nous trouvons différents types de contraintes pouvant être définies explicitement à l'aide du SGDB.

Bien entendu, chacune d'elles peut être gérées de manière implicite dans le système d'information.

En voici une brève énumération :

- *3.1.4.1 la contrainte d'existence.*

  C'est une contrainte d'obligation de valeur, elle exige pour l'attribut qui en est pourvu qu'une valeur lui soit associée.

  La clause NOT NULL permet d'imposer une contrainte d'existence.

```
CREATE TABLE Personnes
(Nom CHAR(20) NOT NULL,
Prénom CHAR(20),
PRIMARY KEY(Nom))
```

Figure 3-5 Clause NOT NULL définissant *la contrainte d'existence.*

D'autres contraintes sont définies en rapport à la présence de cette valeur de nullité.

Dans le modèle Entité/Association étendu on retrouve 4 contraintes spécifiques liées à la présence de la valeur de nullité. Elles sont définies au sein d'un groupe[25].

a) La contrainte de coexistence[26].
« *Les composants du groupe doivent être simultanément présents ou absents pour les instances du parent. La coexistence est représentée graphiquement par le symbole coex. Le parent du groupe est soit un type d'entité, soit un type d'association et les composants sont tous facultatifs* ».

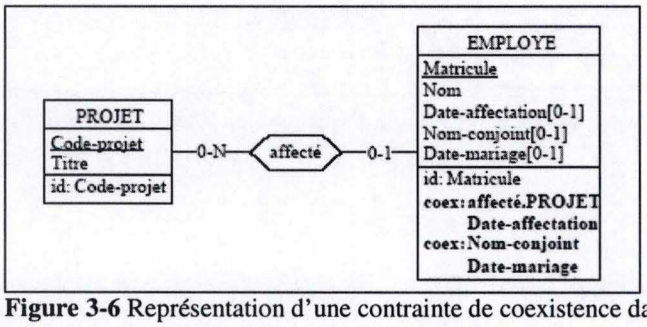


Figure 3-6 Représentation d'une contrainte de coexistence dans DB-MAIN.

b) La contrainte d'exclusivité.
« *Parmi les composants du groupe, au maximum un doit être présent pour chaque instance du parent. Le groupe apparaît graphiquement avec le symbole excl. Le parent du groupe est soit un type d'entité, soit un type d'association et les composants sont tous facultatifs.* ».

[25] « *Un groupe est une collection d'attributs, de rôles et/ou d'autres groupes. Il représente une construction attachée à un parent (un type d'entités, un type d'associations ou un attribut décomposable multivalué) qui est utilisée pour modéliser des contraintes* » [Hick 2001].
[26] Les définitions de ces contraintes sont issues de [Hick 2001].

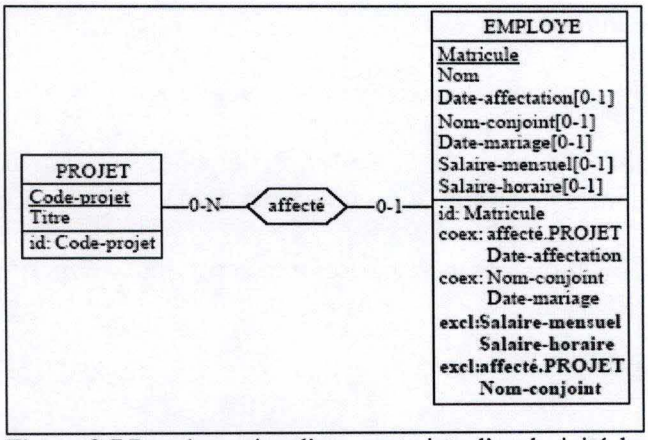


Figure 3-7 Représentation d'une contrainte d'exclusivité dans DB-MAIN.

c) La contrainte au-moins-un.
« Parmi les composants du groupe, au moins un doit être présent pour chaque instance du parent. Le groupe apparaît graphiquement avec le symbole at-lst-1. Le parent du groupe est soit un type d'entité, soit un type d'association et les composants sont tous facultatifs ».

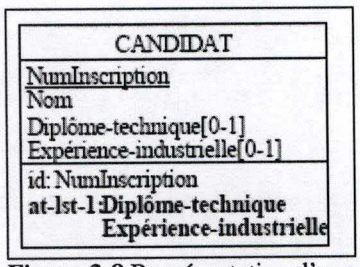


Figure 3-8 Représentation d'une contrainte au-moins-un dans DB-MAIN.

d) La contrainte exactement-un.
« Parmi les composants du groupe, un et un seul doit être présent pour chaque instance du parent (exclusivité et au-moins-un). Le groupe apparaît graphiquement avec le symbole exact-1. Le parent du groupe est soit un type d'entité, soit un type d'association et les composants sont tous facultatifs ».

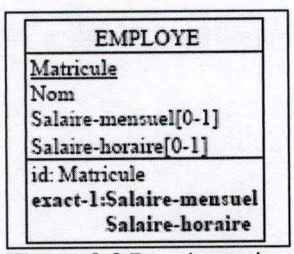


Figure 3-9 Représentation d'une contrainte exactement-un dans DB-MAIN.

➢ *3.1.4.2 la contrainte d'unicité.*

La contrainte d'unicité exige que toute donnée ayant une valeur soit distincte.

Cependant, celle-ci n'oblige pas à ce que les données participant à la formation de la contrainte aient une valeur (la valeur nulle est donc permise).

La clause UNIQUE (<composants>) permet d'imposer une contrainte d'unicité.

```
CREATE TABLE Personnes
(Nom CHAR(20) UNIQUE,
 Prénom CHAR(20))
```

Figure 3-10 Clause UNIQUE définissant la contrainte d'unicité.

➢ *3.1.4.3 la contrainte de la clé primaire.*

C'est une contrainte qui permet d'identifier de manière unique un enregistrement dans une table. Elle regroupe la contrainte d'existence et d'unicité.

Un SGBD relationnel doit donc automatiquement vérifier l'unicité de la clé primaire et son caractère non nul.

La clause PRIMARY KEY (<composants>) permet de définir un(des) champ(s) pouvant servir de clé primaire.

La figure 3-11 montre la représentation d'une clé primaire (en DB-MAIN) sur l'attribut Ncli ainsi que le code LDD permettant de la définir dans la base.

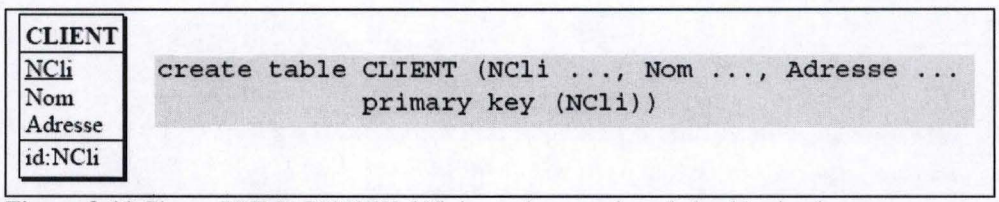


Figure 3-11 Clause PRIMARY KEY définissant la contrainte de la clé primaire.

➢ *3.1.4.4 la contrainte d'intégrité référentielle[27] ou contrainte de clé étrangère.*

Cette contrainte est destinée à assurer l'intégrité de référence servant à empêcher qu'une ligne d'une table qui référence une ligne d'une autre table voit le lien

[27] Comme le lecteur a déjà pu le remarquer, quand nous évoquons le terme contrainte référentielle, nous parlons de clé étrangère.

logique entre les deux lignes brisé. Elle permet ainsi d'établir des liens entre plusieurs tables.

« *Une clé étrangère identifie une colonne ou un ensemble de colonnes d'une table comme référençant une colonne ou un ensemble de colonnes d'une autre table (la table référencée). Les colonnes de la table référencée doivent faire partie d'une contrainte de clé primaire ou d'une contrainte d'unicité. La contrainte de clé étrangère garantit que les valeurs de chaque ligne de la table référençant existent dans la table référencée: ainsi une ligne de la table référençant ne peut pas contenir un ensemble de valeurs qui n'existe pas dans la table référencée* » [Wikipedia].

La clause FOREIGN KEY (<composants>) REFERENCES permet de définir une(des) clé(s) étrangère(s) entre deux tables.

La figure 3-12 montre la représentation (en DB-MAIN) d'une clé étrangère entre l'attribut NCli de COMMANDE et l'attribut Ncli de CLIENT ainsi que le code LDD permettant de la définir dans la base.

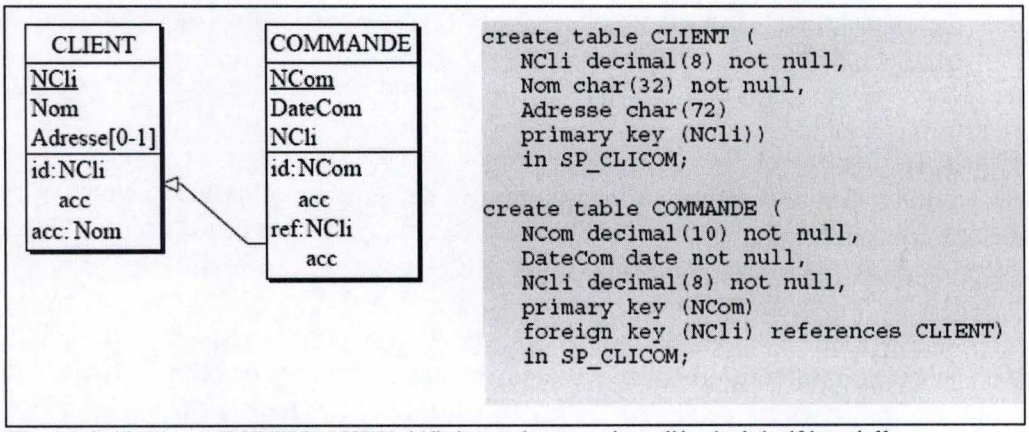


Figure 3-12 Clause FOREIGN KEY définissant la contrainte d'intégrité référentielle.

➢ *3.1.4.5 la contrainte de domaine de l'attribut.*

Les contraintes de domaine sont des contraintes qui restreignent les valeurs qu'une donnée peut prendre.

Par exemple, un domaine POURCENT se verra doté d'une contrainte qui ne lui permet que des valeurs comprises entre 0 et 100.

Le prédicat CHECK (VALUE ...) permet de gérer la contrainte de domaine.

```
CREATE DOMAIN POURCENT FLOAT
CHECK (VALUE BETWEEN 0 AND 100)
```

Figure 3-13 Clause CHECK définissant une contrainte de domaine de l'attribut.

Comme vous avez pu le constater, la variété de contraintes existantes est très large. Explorer et étudier des techniques d'élicitation pour chacune d'elles prendrait beaucoup de temps.

C'est pourquoi nous nous pencherons principalement sur les contraintes d'intégrité référentielle et sur la découverte des clés primaires.
Celles-ci tenteront d'être mises à jour via nos techniques d'analyse présentées dans les chapitres 4 ,5 et 6.

3.2 A la recherche d'une méthode d'élicitation ...

Nous le savons, la rétro-ingénierie est une activité complexe basée sur des méthodes, stratégies, techniques et rôles bien définis [Hainaut 1991].
A travers la littérature, nous avons découvert des approches multiples et variées, explorant diverses solutions aux travers des différentes sources d'information.
Or, la recherche des constructions implicites dans ses nombreuses sources d'information est une tâche hasardeuse pour laquelle il n'existe pas encore de méthodes déterministes.

Cette tâche est bien plus complexe qu'une simple analyse du langage de description de données.

Vu la grande quantité d'information à manipuler, tenter une recherche exhaustive sur toutes les contraintes possibles est irréaliste. Nous avons besoin d'une méthodologie qui nous guide dans nos investigations.

Ce chapitre introduira de ce fait une méthode générique pour la rétro-ingénierie des bases de données relationnelles, consacrée à l'analyse des problèmes de reconnaissance des contraintes implicites.

Mais avant de débuter un projet de rétro-ingénierie, il nous faut définir une méthodologie de travail, fixer les objectifs, définir des critères de recherche, identifier les sources d'information, préciser la nature et l'exactitude des résultats et en citer les risques.

Il est très important de définir et de préciser les résultats attendus avec le client à chaque étape car pour des raisons de coûts, il serait très difficile de faire marche arrière. Il est donc important de l'impliquer même bien avant le début de nos phases d'analyse [Henrard 2002].

Les types de contraintes recherchées, les sources d'information ainsi que les outils ou techniques utilisés pour retrouver les constructions implicites seront les critères sur lesquels nous nous baserons pour établir le processus de recherche.

CHAPITRE 3: METHODOLOGIE DE L'APPROCHE

Notre méthode sera modulaire, opérant à partir de plusieurs sources.
Cette méthode divisée en différentes étapes tentera d'analyser, d'extraire et d'exploiter les traces de contraintes implicites noyées dans le schéma, les données et le code source.

L'exploration de chacune des sources nécessite une technique d'analyse particulière. Cette technique peut être très simple, comme une inspection visuelle, ou être plus complexe, comme une analyse automatique statique ou dynamique.

C 'est pourquoi, trois importants processus d'analyse seront présentés, à savoir : l'analyse de schéma (chapitre 4), l'analyse des données (chapitre 5) et l'analyse de patterns (chapitre 6).

Ceux-ci seront ensuite imbriqués pour éliminer les hypothèses erronées.

A noter que l'exécution de ces activités sera réalisée et présentée en séquence. Cependant, le processus n'est pas pour autant un modèle linéaire, les différentes phases peuvent éventuellement s'entrelacer et être réitérées.

Toutes ces techniques sont complémentaires, elles se combinent et se coordonnent entre elles pour plus d'efficacité, elles ont toutes leur propre particularité et aucune d'elles ne remplacent ou ne surclassent l'autre.

Aussi efficace soit elle, une seule technique ou source d'information ne peut permettre de récupérer toute la sémantique. La recherche de constructeurs ne peut donc se limiter à l'analyse d'une seule source. La force de cette méthodologie est donc de pouvoir extraire les informations provenant de sources très variées comme le code, le schéma ou les données même s'il en existe d'autres comme les écrans, les rapports, la documentation, les interviews, etc.

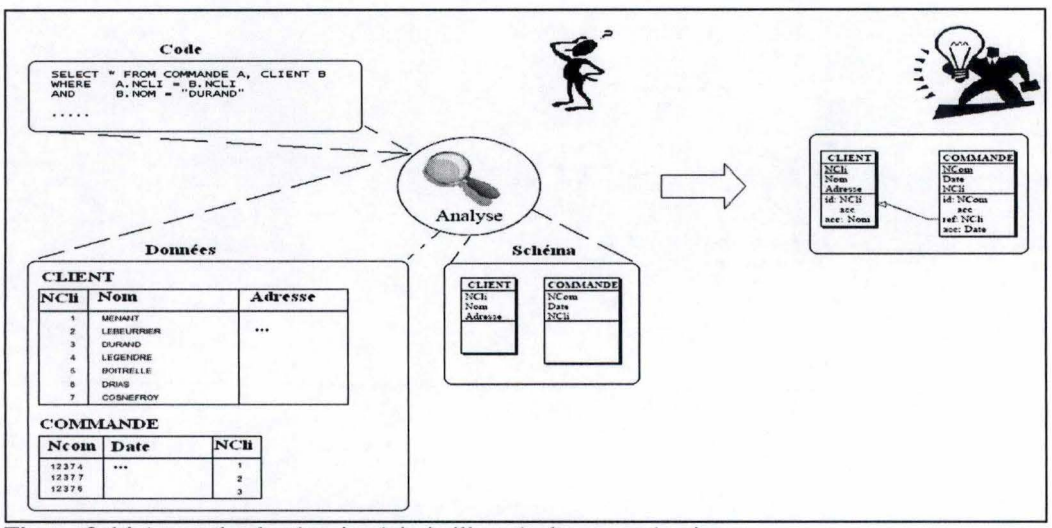


Figure 3-14 Approche de rétro-ingénierie illustrée dans ce mémoire.

3.3 Méthodologie de rétro-ingénierie

En partant de l'hypothèse que le schéma se trouve en 3FN, la première étape du processus se consacre donc à l'extraction et à la détection d'indicateurs dans le schéma physique relationnel pour partir à la recherche de clés primaires et étrangères implicites.

Ensuite, en partant des hypothèses formulées sur base de l'analyse du schéma, l'analyse des données de la base qui lui est associé nous permet d'obtenir des indicateurs plus pertinents et plus significatifs pour valider et vérifier l'existence des contraintes implicites potentielles identifiées par le schéma.

La troisième étape, l'analyse de patterns, apportera une couche supplémentaire dans la validation des hypothèses de départ. En effet, elle consiste à analyser les programmes applicatifs à la recherche de patterns pouvant mettre l'accent sur d'éventuels indices. Indices qui infirmeront ou confirmeront ces hypothèses de départ et pourront en solliciter de nouvelles.

Il appartient alors au rétro-concepteur de les confronter afin de valider ou de rejeter toutes les hypothèses émises à partir des différentes sources d'information.
Celui-ci procèdera ensuite à l'étape d'enrichissement du schéma sur base des contraintes implicites potentielles ayant été élicitées.

Le processus se terminera par l'étape d'amélioration de la qualité du système d'information (chapitre 8).

Pour nous aider au travers de ces étapes, des plugins ont été développés. Ceux-ci se sont avérés nécessaires pour accompagner nos phases d'analyse.

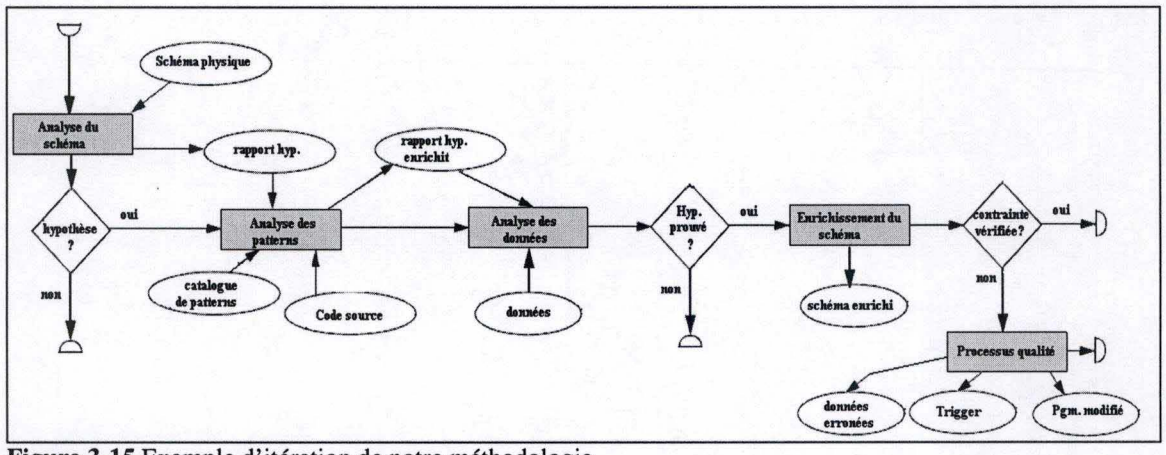


Figure 3-15 Exemple d'itération de notre méthodologie.

3.4 Pourquoi le besoin d'une troisième forme normale ?

Beaucoup d'approches émettent l'hypothèse que le schéma se trouve en troisième forme normale ([Navathe 1988], [Davis 1988], [Chiang 1993], [Johannesson 1994]).

Ce besoin de normalisation en 3FN n'apparaît pas comme un aspect contraignant pour notre processus d'analyse. En effet, il existe des algorithmes pouvant être utilisés pour normaliser le schéma [Britton 1989] et les détecter.

La raison principale de ce besoin pour commencer notre processus est due au fait que les relations en troisième forme normale représentent d'une bonne manière les structures des objets dans le monde des bases de données. Elles éliminent la redondance des données et assurent une meilleure intégrité.

La quatrième forme normale (4FN)[28] et la cinquième forme normale (5FN) sont rarement prises en compte en pratique. Elles s'éloignent trop de la structure originale des objets. Quant à la première (1FN) et la deuxième forme normale (2FN), elles contiennent trop d'informations redondantes que pour être analysées facilement.

3.5 Automatisation de la méthodologie

Chaque processus a un degré d'automatisation, certains le sont plus facilement que d'autres mais aucun ne l'est à 100%.

L'automatisation complète des différentes techniques est pratiquement impossible car les informations récupérées doivent être consolidées et validées par un ou plusieurs ingénieurs ayant la connaissance du domaine de l'application. C'est lui qui sera en charge de résoudre les conflits; son rôle est donc très important. C'est clairement une activité basée sur le décisionnel ce qui rend l'automatisme difficile.
Cette interaction est impérative pour capturer l'essentiel de la sémantique. L'utilisateur doit rester au cœur du système, car c'est via cette interaction qu'il peut confirmer les hypothèses proposées ou formuler une alternative.

Une autre raison de la difficulté de cette « full » automatisation est que chaque projet est différent et a ses propres caractéristiques. L'ingénieur doit donc définir les outils, les méthodes et les approches qu'il va employer avant de commencer le projet.

Néanmoins, l'automatisation même partielle est inévitable car elle permet d'obtenir des indices et donc des résultats plus rapidement.

[28] Egalement nommée forme normale de Boyce-Codd (FNBC).

3.5.1 Besoin d'outils

Afin d'accompagner les différentes techniques d'analyses, nous avons développés divers plugins sans lesquels aucun résultat probant n'aurait pu être mis en valeur.
Le rôle principal de ces plugins est d'augmenter la productivité et d'attirer avant tout l'attention de l'ingénieur.

3.5.2 Importance et rôle de la connaissance humaine

C'est dans la connaissance humaine que réside toute la limitation d'un processus automatique, car pour interpréter les résultats dans le domaine de la rétro-ingénierie, plus qu'ailleurs, nous avons besoin d'un analyste.

N'oublions pas que l'humain détient généralement une bonne partie des connaissances du domaine et de l'application; c'est même une source d'information de grande valeur.
Celui-ci occupe donc un rôle important dans la prise de décisions, mais également dans l'enrichissement des informations. Son interaction et sa présence sont donc primordiale dans les processus de rétro-ingénierie.
Celui-ci doit non seulement maîtriser les techniques et les outils d'analyse mais doit également avoir une grande connaissance du domaine de l'application. Sans cela, il pourrait commettre des erreurs d'interprétation de résultats et produire des schémas incorrects.

Par cette phrase, reprise ci-dessus, nous pouvons classer notre approche dans les méthodes dites « semi-automatisé », où l'ingénieur sera toujours présent et apportera son savoir et son expérience dans la prise de décisions.
Son intervention est nécessaire pour traiter les instances problématiques. En effet, l'automatisation peut produire des résultats erronés, non précis ou incomplets.

Dans la rétro-ingénierie, la « full automatisation » dans une méthodologie est assez rare pour ne pas dire presque inexistante. Chaque projet a ses propres caractéristiques et sa propre complexité, cela n'empêche pas certaines activités de l'approche d'être partiellement ou complètement automatisées.

Il faut également garder à l'esprit le challenge économique. Nous somme actuellement dans une politique de réduction de coût et aucune entreprise, à l'heure actuelle, ne choisira une solution 100% manuelle en raison de son coût trop élevé. Il faut donc s'assurer que les projets de rétro-ingénierie soient réalisés dans des temps raisonnables et à des coûts acceptables [Henrard 2000].

3.6 Fin du processus

Le processus final sera manuel; il visera avant tout à enrichir le schéma de départ de la base de données. Les objets cachés (contraintes implicites) seront alors mis à jour grâce à l'analyse, à la confrontation et à la validation des hypothèses issues des différentes sources d'information disponibles (chapitre 7).

Les hypothèses validées seront alors converties en constructions physiques (code LDD, triggers, code applicatif) et le schéma de départ sera enrichi.

C'est l'ingénieur qui définira la fin du processus en fonction de la complétude des résultats obtenu et des coûts du projet.

En effet, il est impossible de savoir quand toutes les contraintes implicites seront découvertes, car nous ne disposons pas du schéma de référence auquel nous pouvons comparer nos résultats.

3.7 Limites

Il existe une grande variété de contraintes pouvant être présentes dans les constructeurs implicites. Celles que nous regarderons et que nous analyserons sont principalement les contraintes référentielles et les identifiants.

Dans ce mémoire, nous avons pris l'option de convertir les clés étrangères en relations simples, leurs interprétations se feront dans la phase de normalisation.

Ce processus n'est pas sans failles, certains conflits ou inconsistances peuvent survenir. En effet des erreurs peuvent être présentes dans les programmes, dans les données ou dans le schéma. L'ingénieur peut également faire des erreurs au moment de la phase de validation des hypothèses.

Le résultat final sera bien souvent incomplet car il y aura toujours une partie de code, une partie du schéma ou une partie des données qui restera inexplorée.

3.8 Vers des constructions plus riches

Dans un schéma conceptuel, d'autres structures bien plus complexes que les clés étrangères sont représentées comme les attributs multivalués et/ou décomposables, les types d'associations ou les relations « IS-A » (figure 3-16).

Celles-ci n'ont pas d'équivalence dans le schéma physique relationnel. Elles sont transformées lors de la conception logique pour correspondre à des structures qui sont directement traduisibles en instructions SQL (conception physique). Vu sa simplicité, le modèle relationnel ne peut exprimer ce genre de représentation abstraite.

En effet, pour correspondre au modèle de représentation du schéma relationnel, ces contraintes ne peuvent figurer dans le schéma physique.

Même si nous nous limitons à la recherche des clés primaires et des clés étrangères, celles-ci peuvent révéler des traces pouvant conduire à la découverte d'autres constructions de conception plus riches comme l'héritage ou la définition des cardinalités qui sont définies au niveau du schéma conceptuel.

Par cette section, nous voulons juste montrer que nos méthodes d'analyses peuvent également fournir des indices pouvant servir lors des phases de normalisation et des phases de transformation du schéma physique vers le schéma conceptuel.

Rappelons que contrairement au modèle Entité/Association et au modèle orienté objet, le modèle relationnel ne dispose pas du concept généralisation/spécialisation (héritage IS-A dans le modèle objet). Dans [Rama 1997], [Akoka 1999] et [Lammari 1999] des techniques de rétro-conception de hiérarchie de généralisation/spécialisation ont été décrites.

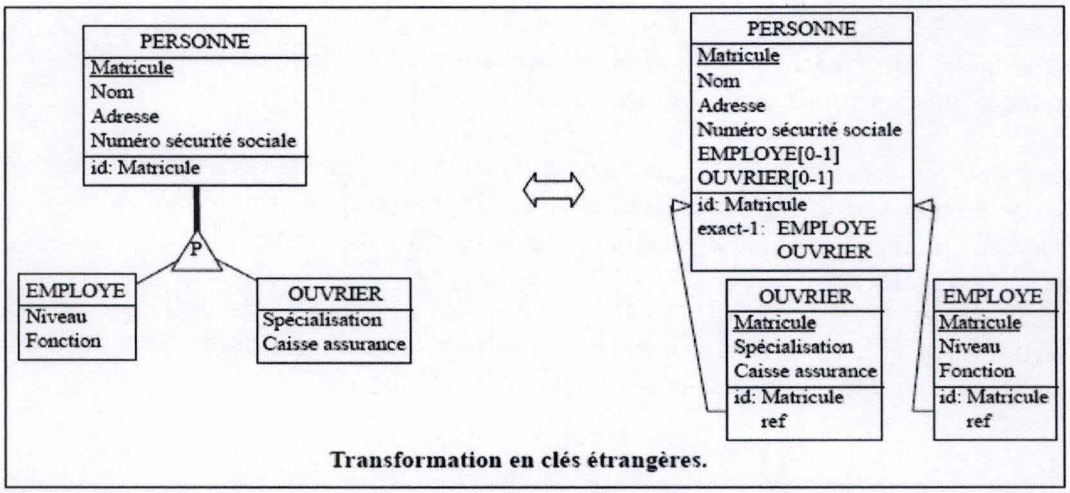


Figure 3-16 Transformation au niveau logique d'une relation IS-A en clés étrangères [Hainaut 2006].

3.9 Conclusion

Atteindre un résultat dit complet serait utopique, mais nous tenterons de nous en approcher un maximum et de fournir un résultat riche et correct.

En effet, contrairement au processus de conception qui prend fin dès que toutes les fonctionnalités du système ont été correctement implémentées, le processus de rétro-ingénierie ne peut se reposer sur ce genre de critères.
En effet, chaque nouvelle analyse provenant de sources et outils divers est susceptible de mettre à jour de nouvelles contraintes implicites et donc de fournir un résultat plus étoffé.

Dans la rétro-ingénierie, il existe autant d'approches qu'il y a de projets, il serait donc illusoire d'en définir une à suivre.
C'est pourquoi, la démarche exposée dans ce mémoire a précisément pour objectif de proposer une approche illustrant différents domaines, outils et techniques de cette discipline.

Le processus songera également à faire évoluer la qualité du système en renforçant le respect des contraintes identifiées tant du côté de la base de données que du côté applicatif (chapitre 8).

Nous présenterons dans les chapitres suivants un (des) exemple(s) très réduit(s) et simple(s) à chacune des phases, mais représentatif et clair, illustrant l'importance de chacune de nos techniques d'analyse.

Chapitre 4

ANALYSE DU SCHEMA

4.1 Introduction

Comme évoqué dans la description de notre méthodologie, une des sources d'information que nous allons analyser est le schéma physique relationnel. Celui-ci comporte à la fois des constructions explicites et implicites et constitue un bon point de départ pour débuter notre approche.

Il est souvent plus judicieux de se concentrer d'abord sur celui-ci plutôt que d'essayer de parcourir tous les composants de l'application.

Bien qu'en règle générale, le schéma soit une source d'information assez complète au départ, il n'empêche que certaines informations essentielles peuvent manquer. Il devra donc être vérifié et enrichi au moyen de notre analyseur de schéma et par l'analyse d'autres composants de l'application.

Ce premier processus montrera que certaines contraintes implicites peuvent être élicitées grâce à l'analyse du schéma physique de la base de données relationnelle.

Rappelons qu'il existe trois niveaux de modélisation pour la représentation des systèmes d'information : conceptuel, logique et physique. Comme expliqué précédemment, notre analyse ne tiendra compte que du modèle physique mais une brève explication des différents niveaux de représentation du modèle des données sera introduite dans ce chapitre.

Pour construire les bases de notre approche d'analyse du schéma, nous nous sommes inspirés d'un outil d'assistance pour l'élicitation de clés étrangères existant dans DB-MAIN. Cet outil appelé « the foreign key discovery assistant »[29] part d'un groupe de champs cible (ou origine) d'une clé étrangère et recherche dans le schéma tous les groupes de champs qui peuvent être origine (ou cible) de ce groupe en se basant sur les critères encodés pour le « matching » tel que la longueur des champs, le type des champs et les règles de comparaison des noms des champs [Henrard 2002].

Notre approche tentera d'apporter une touche innovatrice de par son interface et sa présentation des résultats, tout en étoffant certaines heuristiques pour la détection des identifiants et des clés étrangères implicites, en y rajoutant notamment un comparateur de noms utilisant un indicateur de ressemblance basé sur l'algorithme de reconnaissance de

[29] Un aperçu de l'outil est donné dans [Henrard 2003].

White [White 2004], un analyseur de préfixes et de suffixes, un dictionnaire de synonymes et un traducteur pour le nom des attributs.

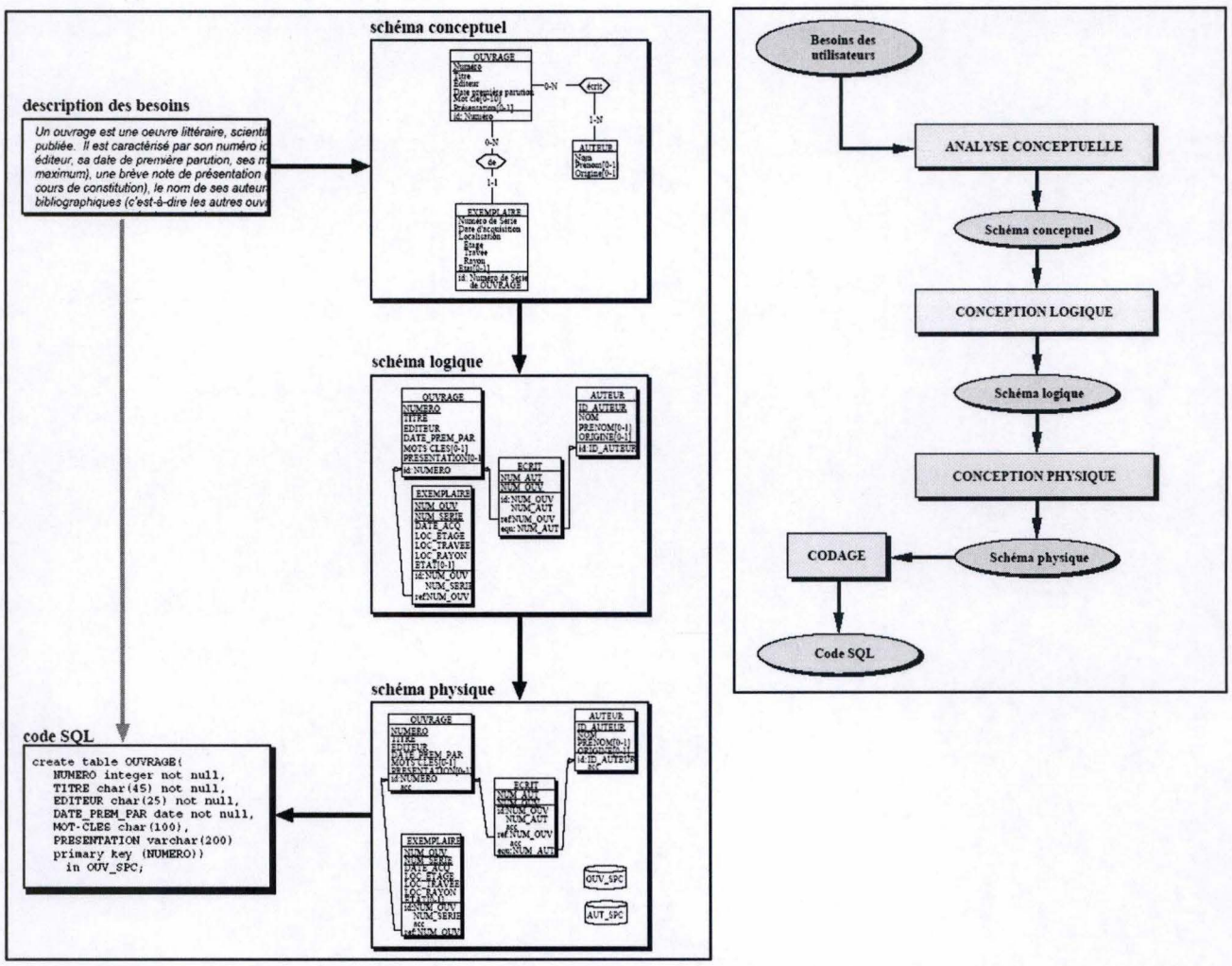
L'approche peut présenter certains avantages. En effet, les types de données spécifiés dans le langage LDD sont assez simples et peu nombreux. Cependant l'analyse statique du schéma peut également présenter certaines faiblesses car basée sur des hypothèses assez fortes, comme le fait que le schéma est supposé être au minimum en troisième forme normale et que le concepteur est resté cohérent dans le nommage des attributs de la base de données.

Notre domaine de recherche sera concentré sur la détection des clés primaires et des clés étrangères potentielles présentes dans le schéma.

4.2 Vue d'ensemble des différents niveaux d'abstraction

Dans un projet de rétro-ingénierie il est couramment admis qu'il existe trois niveaux de représentation, à savoir : le niveau conceptuel, le niveau logique et le niveau physique ([Tardieu 1983], [Teo 1989], [Hainaut 2002]).

Parler de ces trois niveaux d'abstraction est très important pour pouvoir bien comprendre et mieux appréhender les processus de conception et de rétro-conception des bases de données (figures 4-1).



Figures 4-1 Processus standard de conception des bases de données [Hainaut 2002].

4.2.1 Niveau conceptuel

La phase d'analyse conceptuelle a pour but de transformer en schéma conceptuel l'expression formelle et abstraite des besoins des utilisateurs.

Elle s'appuie sur une analyse de base par laquelle les sources d'information (non formelles ou semi formelles) sont analysées et leur contenu sémantique est traduit en structures conceptuelles.
La normalisation permet de donner à ces structures des qualités telles que la lisibilité, la normalité et la conformité aux normes de représentation.

L'analyse conceptuelle est généralement formalisée sous la forme de schémas Entité/Association.

4.2.2 Niveau logique

L'objectif de cette phase de conception est de produire un schéma équivalent au schéma conceptuel en ce qui concerne le contenu informationnel tout en restant conforme au modèle logique. Le schéma logique répond à des besoins techniques et organisationnels [Hick 2001].

Dans le processus de conception logique, l'objectif est l'optimisation qui mène à l'amélioration de la performance.

A noter aussi, qu'à partir de ce schéma logique, on dérive des vues qui répondent aux besoins de chaque utilisateur.

4.2.3 Niveau physique

Dans la phase de conception physique, le schéma logique est enrichi avec des caractéristiques techniques spécifiques du SGBD pour créer le schéma physique.

Au final, le schéma physique et les vues des utilisateurs sont convertis en code opérationnel.

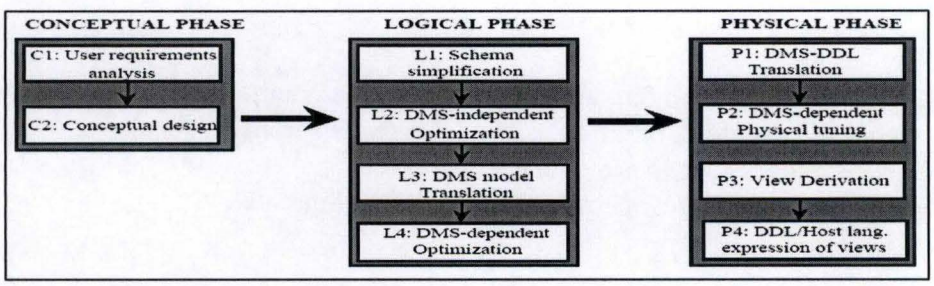


Figure 4-2 Phases importantes du processus de conception des bases de données [Hainaut 1991].

4.3 Approche

4.3.1 Input/output

Le schéma physique relationnel sera l'input de notre analyseur. Dans la rétro-ingénierie celui-ci est généralement obtenu après la réalisation du processus d'analyse du code LDD.

En output, nous retrouverons un schéma enrichi et affiné manuellement par l'ingénieur en fonction des résultats du rapport issus de notre analyseur de schéma.

Rappelons qu'il s'agit d'un résultat « partiel ». En effet, une analyse plus approfondie d'autres composants de l'application va suivre ce processus.

4.3.2 Principes généraux

Le processus général repose sur l'analyse du schéma physique en 3FN.

Pour décider quel champ est susceptible d'être un identifiant potentiel et afin de réduire l'étendue des recherches, l'approche se basera sur la nomenclature du nom des attributs (contenant des mots clés comme « id », « num » et « code »), sur la similitude entre le nom de l'attribut et celui de son entité, sur son type et sur sa position dans la table.

Pour retrouver les clés étrangères implicites, nous proposons de partir d'une idée simple. Afin de trouver une corrélation entre les attributs, nous allons comparer les caractéristiques des champs entre eux à la recherche de similitudes.
Les heuristiques principales pour la recherche des clés étrangères implicites sont basées sur l'examen du nom des attributs (pourcentage de ressemblance, synonymes, traduction). En effet, le nom d'un champ ou d'une table de référence contient souvent sous une forme plus ou moins explicite des informations le concernant et concernant sa cible. Les programmeurs expérimentés font attention à donner des noms cohérents (« naming convention »[30]) lors de la conception de la base de données.

Mais, outre leur nom, beaucoup d'autres indicateurs peuvent et doivent jouer un rôle important dans la découverte d'indices sur la sémantique de l'application tel que leur type et leur taille, ce qui aura pour but d'éliminer les inconsistances liées à la présence d'homonymes.

Toutes ces heuristiques nous semblent être un bon point d'entrée pour débuter la recherche des contraintes implicites à partir du schéma. Celles-ci seront présentées et tenteront d'être vérifiées dans les sections suivantes.
Attention, le schéma peut présenter des incohérences ou des ambiguïtés.

[30] Convention de nommage.

En effet, des champs similaires peuvent présenter des noms différents et même être décrits avec des types différents. Il devient donc difficile de mettre en relation des attributs ayant des noms non consistants et ne respectant pas une terminologie bien définie.

C'est pourquoi un rapprochement ou un éloignement de ceux-ci par l'utilisation d'un dictionnaire de synonymes et de traduction des noms[31] tentera de donner plus d'innovation et de cohérence à cette approche.

Notons que si trop de règles restrictives sont introduites pour le « matching », l'outil générera du « silence » (passera à côté de clés étrangères ou primaires existantes). A l'inverse, si les règles sont trop vagues du « bruit » sera généré (suggestion de clés erronées).
Le bruit peut-être facilement éliminé par l'utilisation d'autres techniques d'analyse.
Par contre le silence est plus complexe à détecter.

In fine, un rapport reprenant toutes les clés primaires et étrangères possibles sera produit afin d'être vérifié et validé par l'ingénieur.

A ce stade le résultat obtenu n'est encore que « partiel » car l'information extraite des données et des patterns provenant du code source doit encore y être intégrée.
Celle-ci viendra compléter et enrichir, au fur et à mesure, le résultat obtenu par l'analyse du schéma.

Le but final est de fournir des indices pouvant mener à un schéma physique raffiné.

Même si certains noms mériteraient d'être rendus plus expressifs. Nous ne procèderons toutefois pas à la substitution des noms des champs en leur donnant des noms plus significatifs ou plus expressifs. Nous conserverons les noms d'origine, ceux-ci pouvant être utilisés pour la maintenance ou le développement.

En raison du temps imparti et de la charge de travail déjà assez conséquente, les auto-jointures[32] ne seront pas prises en considération dans cette analyse.

4.3.3 Automatisation

L'outil mis en place pour l'analyse du schéma détectera automatiquement toutes les clés étrangères et primaires possibles répondant à un ensemble de règles de « matching ».
L'ingénieur sera, quand à lui, impliqué dans les prises de décisions finales, sa responsabilité étant d'accepter ou de réfuter les hypothèses proposées par l'outil.
On parlera donc de procédé semi-automatisé.

---

[31] Beaucoup d'applications sont multilingues, plus particulièrement en Belgique où il existe trois langues légales à savoir le français, le néerlandais et l'allemand. De plus l'anglais est souvent utilisé par les concepteurs.
[32] Jointure d'une table avec elle-même.

4.3.4 Heuristiques

Une règle de bonne pratique est d'utiliser un grands nombre d'heuristiques pour rechercher une hypothèse. Nous en proposons de ce fait quelques unes basées sur des règles de bonne pratique. Leur efficacité sera vérifiée lors des nos diverses illustrations.

Nous classerons nos heuristiques en fonction de leur rôle.

➢ *4.3.4.1 La découverte des clés primaires*

Les recherches se font de la manière suivante :

a) La nomenclature du nom de l'attribut peut contenir les mots clés « ID », « NUM », « CODE » dans son préfixe ou son suffixe. Le périmètre de recherche peut être affiné en définissant la position[33] et le type[34] de l'attribut à analyser.

b) La nomenclature du nom de l'attribut peut être similaire à son entité[35]. Pour définir cette similitude nous avons utilisé l'algorithme illustré dans [White 2004].
Un exemple illustratif est donné dans la figure 4-3.
Le périmètre de recherche peut également être affiné en définissant la position et le type de l'attribut à analyser.

Nom	*Ressemblance avec "Dupond"*
dupond	100%
dupon	88%
dupont	80%
durand	40%
durant	20%
Aline	0%

Figure 4-3 Exemple de résultats obtenus par l'utilisation de l'algorithme de « matching » décrit dans [White 2004].

NB : Pour augmenter le pourcentage de ressemblance lors de la comparaison des noms et des mots clés, tous les noms des objets du schéma ont été mis en majuscule et les espaces blancs ont été supprimés.

c) Les attributs facultatifs seront exclus par notre analyseur de schéma. Ceux-ci ne rentrent pas dans les critères de sélections pour la détection des clés primaires. En effet, la contrainte d'existence est une des conditions sine qua non pour la définition d'une clé primaire.

[33] Généralement un identifiant se trouve dans les premiers champs de la table.
[34] Nous avons considéré uniquement les entiers, les caractères et les dates comme type susceptible d'être utilisé par un attribut identifiant. Ceux-ci semblent être utilisés dans la plupart des SGBD.
[35] Il nous semblait intéressant d'inclure cette heuristique et de la vérifier par la suite. Car dans certains cas, il se peut que le concepteur donne à l'attribut identifiant le même nom que son entité.

➢ *4.3.4.2 La découverte des clés étrangères.*

Ce processus de recherche ne peut commencer que si le schéma contient au moins un identifiant (clé primaire) ou un identifiant secondaire puisque au départ chacun de ceux-ci est susceptible de constituer une clé référencée potentielle.

Les recherches se font de la manière suivante :

a) La nomenclature du nom de l'attribut enfant peut être similaire ou identique à celui de son parent. La similitude est définie par l'algorithme de White.

b) Le nom de l'attribut enfant peut être une traduction de son parent. Celui-ci est défini grâce à notre dictionnaire de traduction.

c) Le nom de l'attribut enfant peut être un synonyme de son parent. Celui-ci est défini grâce à notre dictionnaire de synonymes.

Le périmètre de recherche peut également être affiné en ajoutant des critères de correspondance basés sur le type ou la taille[36].

NB : Pour augmenter le pourcentage de ressemblance et faciliter la recherche dans le dictionnaire de synonymes et de traduction, tous les noms des objets du schéma ont été mis en majuscule et les espaces ont été supprimés. Il nous semblait judicieux d'également supprimer[37] les préfixes « ref-», «ref_», «id-», «id_», «num-», «num_», «code-», «code_ » et les suffixes « -id », « _id », « -num », « _num », «-code», «_code» (figure 4-4). Cette liste n'est pas exhaustive et d'autres préfixes ou suffixes peuvent compléter cette liste.

Nom 1	*Nom 2*	*% Ressemblance entre Nom 1 et Nom 2*	
		avec préfixes et suffixes	*sans préfixes et suffixes*
ref-dient	dient	71%	100%
cust-id	cust	66%	100%
num_dient	dient	71%	100%
cust-id	cust-id	100%	100%

Figure 4-4 Exemple de comparaison de résultats obtenus en supprimant ou en laissant les suffixes et préfixes lors de l'analyse des noms.

[36] Pour plus de souplesse, un plus faible « size checking » peut être défini. Celui-ci autorise la comparaison d'attributs de taille plus petite avec des attributs de taille plus grande. Par exemple, un attribut enfant de type char(3) est compatible avec un attribut parent de type char(4), l'inverse n'est pas autorisé.

[37] Ces mots clés sont souvent utilisés pour désigner un identifiant ou une clé référencée.
Leur suppression permet de rendre plus efficace la comparaison des noms, la recherche des synonymes et la recherche des traductions.

4.4 Illustration

Dans le but d'illustrer nos idées, un plugin d'analyse de schéma a été implémenté à l'aide de l'API de DB-MAIN. Nous présenterons dans cette section plusieurs exemples illustrant l'importance de cette technique d'analyse.

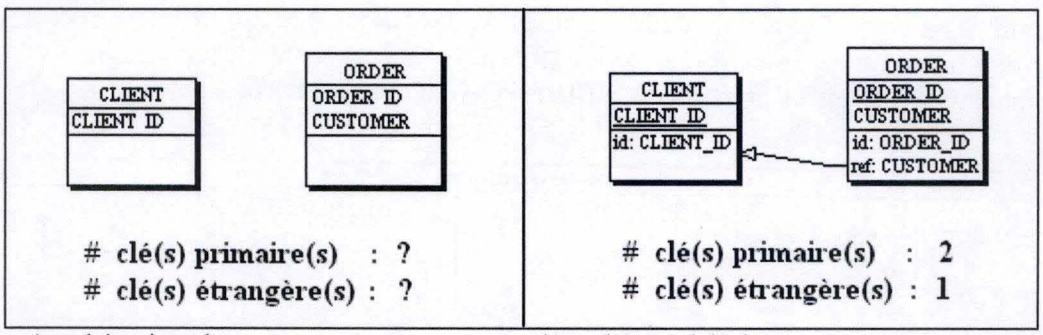
L'API de BD-MAIN et les fonctionnalités de ce plugin ainsi que les rapports générés par notre analyseur seront détaillés en annexe.

Les illustrations ont pour but de présenter toute une série de petits cas concrets illustrant l'importance de certaines techniques de recherche implémentées dans notre analyseur de schéma.

4.4.1 Première illustration

La première illustration a été réalisée sur le schéma physique de la figure 4-5. Cet exemple permet de constater l'utilité du dictionnaire de synonymes. Celui-ci n'a pas été pris au hasard puisque la contrainte référentielle présente dans ce schéma n'avait pas été identifiée par l'outil « the foreign key discovery assistant » de l'atelier de DB-MAIN.

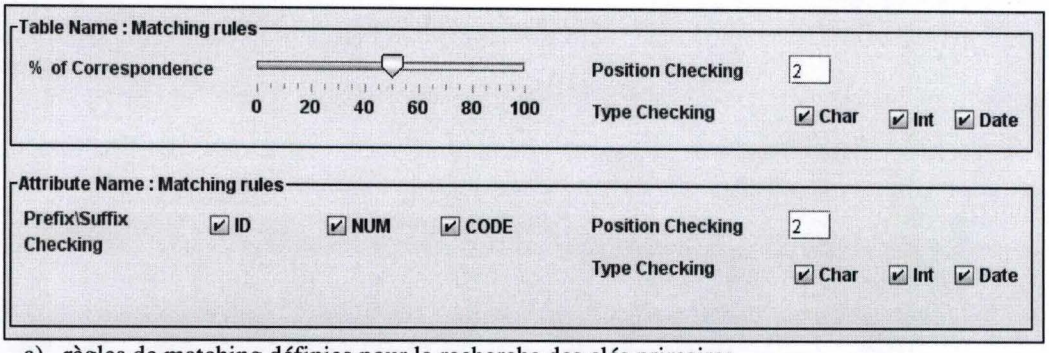
Le dictionnaire de synonymes et le dictionnaire de traduction ont été alimentés en partie pour le besoin de l'exemple. Il est évident que si celui-ci disposait d'une base de données plus complète, la recherche en serait améliorée.



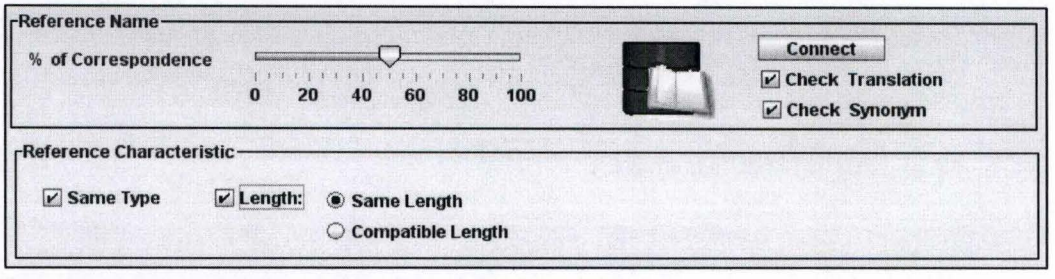
a) schéma à analyser. b) schéma original.

Figure 4-5 Schémas physiques utilisé pour la première illustration.

Pour cette analyse les critères de « matching » ont été définis comme présenté dans les figures 4-6.



a) règles de matching définies pour la recherche des clés primaires.



b) règles de matching définies pour la recherche des clés étrangères.

Figures 4-6 Règles de matching utilisées pour la première illustration.

La figure 4-7 nous montre les heuristiques ayant été utiles pour la détection des différents indices.

	Heuristique	Heuristique Vérifiée
Clé primaire	Correspondance du nom de l'attribut avec entité	
	Préfixe/Suffixe ID	✓
	Préfixe/Suffixe NUM	
	Préfixe/Suffixe CODE	
	Position	✓
	Type de l'attribut (INT)	✓
	Type de l'attribut (CHAR)	
	Type de l'attribut (DATE)	
Clé étrangère	Correspondance des noms	
	Traduction	
	Synonyme	✓
	Type identique	✓
	Taille identique	✓
	Taille compatible	

Figure 4-7 Heuristiques vérifiées pour la première illustration.

Un schéma quantifiant les résultats est présenté dans la figure 4-8.
Les résultats du schéma analysé correspondent parfaitement au schéma original. Aucun bruit, ni silence n'a été généré lors de cette analyse.

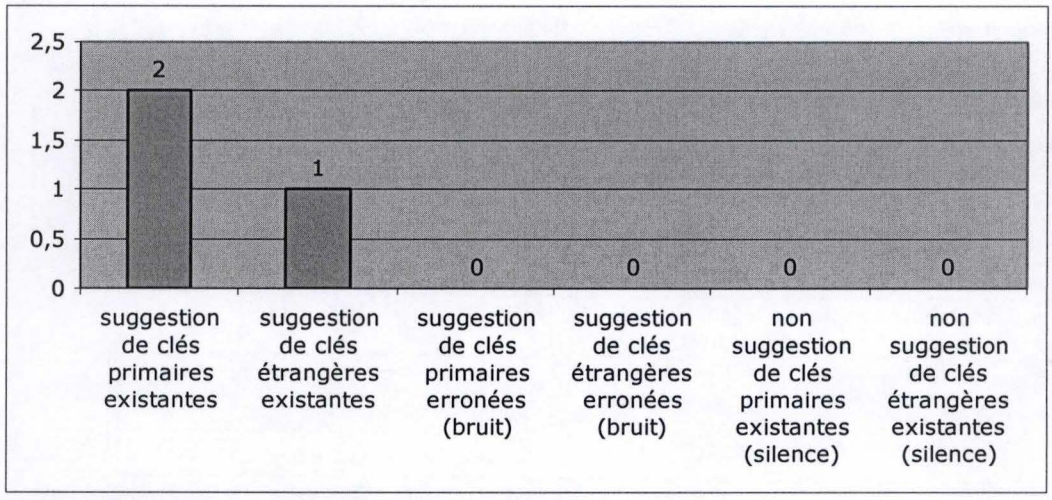
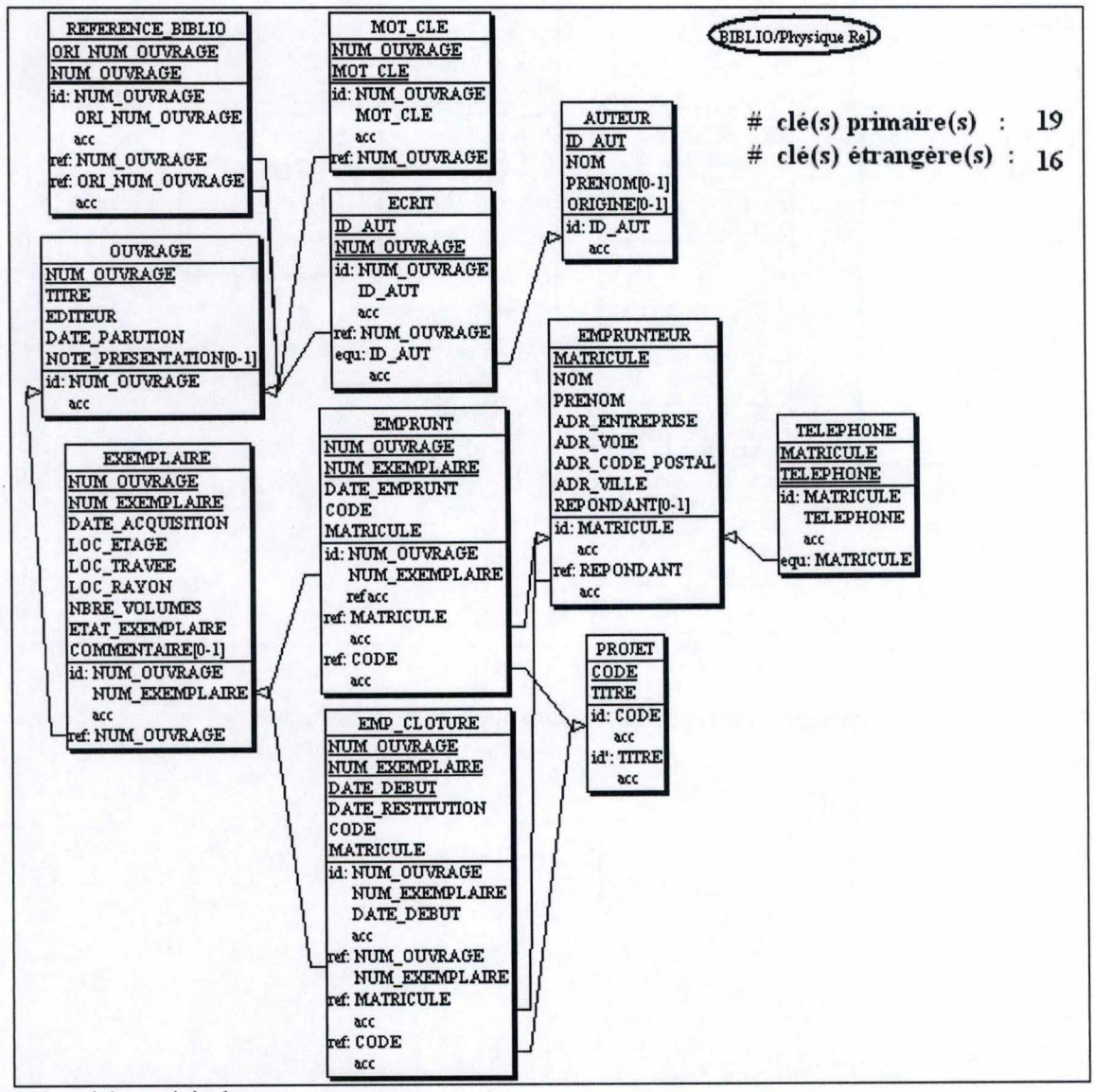


Figure 4-8 Quantification du résultat de la première illustration.

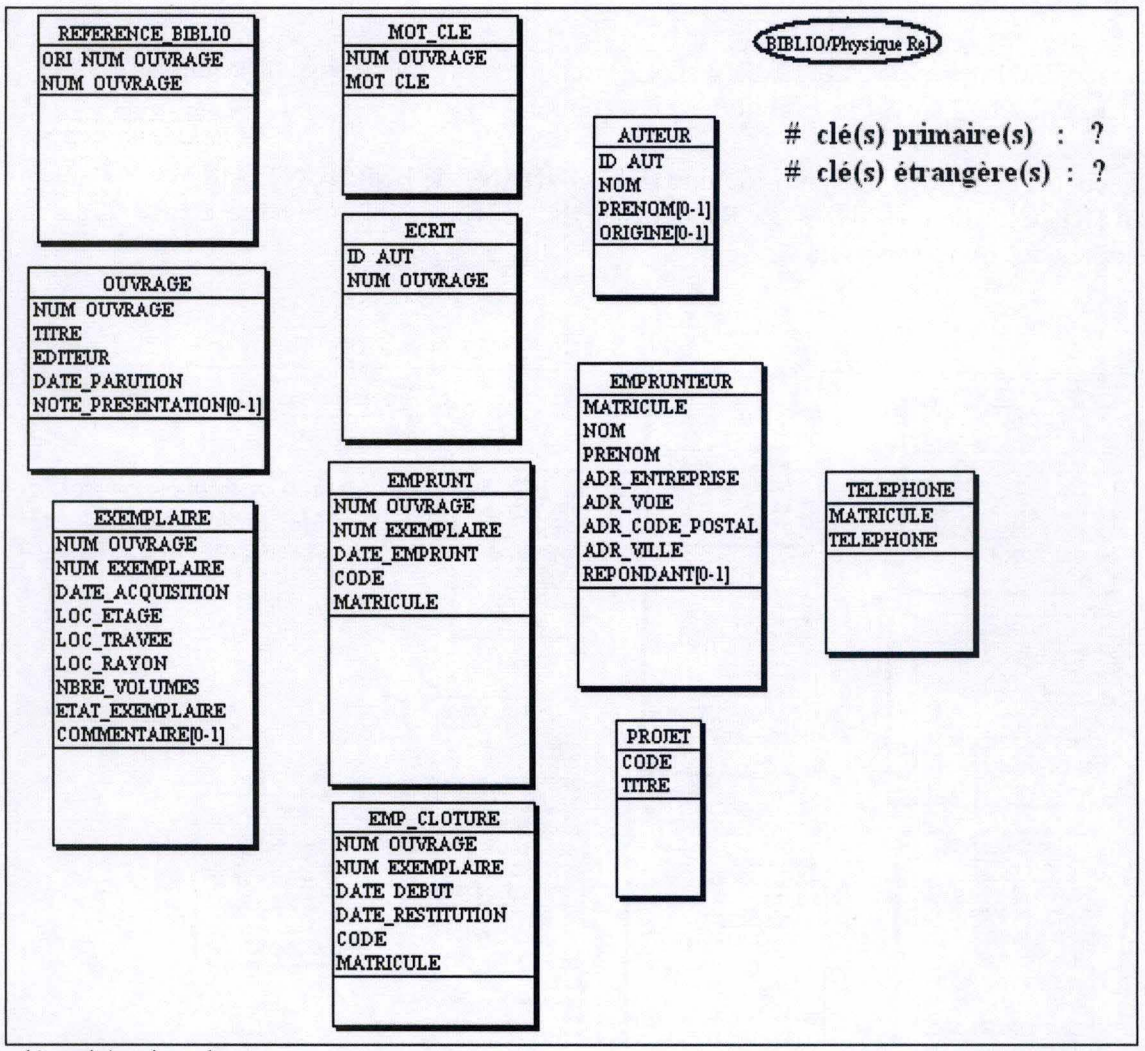
4.4.2 Deuxième illustration

La seconde illustration a été réalisée sur un exemple académique issu du cours de [Hainaut 2006] à savoir la « bibliothèque » (figure 4-9).

Ce schéma comprend une auto-jointure (REPONDANT de EMPRUNTEUR vers MATRICULE de EMPRUNTEUR). Pour rappel, ce type de jointure ne sera pas pris en compte dans notre analyseur.



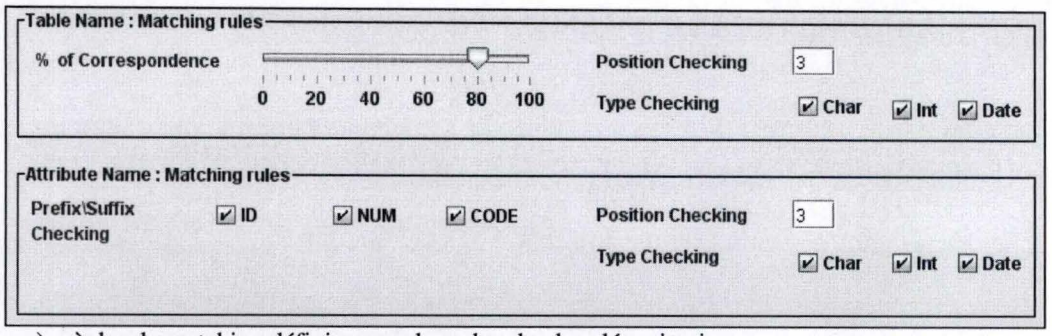
a) schéma original.



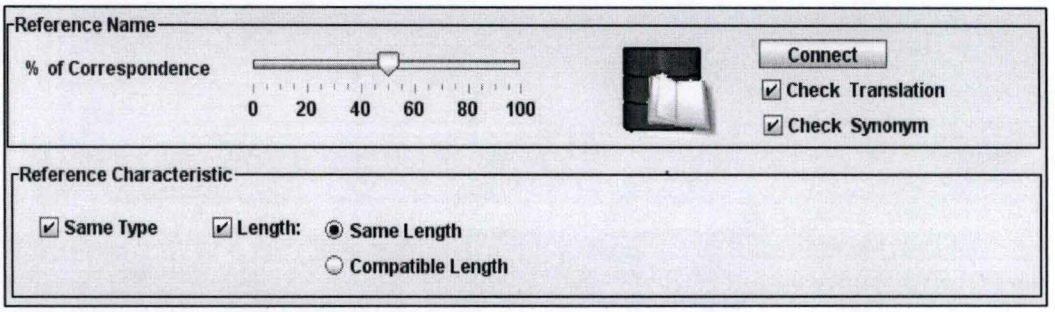
b) schéma à analyser.

Figures 4-9 Schémas physiques utilisés pour la deuxième illustration (Bibliothèque).

Pour cette analyse, les critères de « matching » ont été définis comme présenté dans les figures 4-10.



a) règles de matching définies pour la recherche des clés primaires.



b) règles de matching définies pour la recherche des clés étrangères.

Figures 4-10 Règles de matching utilisés pour la deuxième illustration.

La figure 4-11 nous montre les heuristiques ayant été utiles pour la détection des différents indices.

	Heuristique	Heuristique Vérifiée
Clé primaire	Correspondance du nom de l'attribut avec entité	✓
	Préfixe/Suffixe ID	✓
	Préfixe/Suffixe NUM	✓
	Préfixe/Suffixe CODE	✓
	Position	✓
	Type de l'attribut (INT)	✓
	Type de l'attribut (CHAR)	✓
	Type de l'attribut (DATE)	
Clé étrangère	Correspondance des noms	✓
	Traduction	
	Synonyme	
	Type identique	✓
	Taille identique	✓
	Taille compatible	

Figure 4-11 Heuristiques vérifiées pour la deuxième illustration.

Le schéma quantifiant les résultats est présenté dans la figure 4-12.

Pour la recherche des clés primaires, les clés : ORI_NUM_OUVRAGE de REFERENCE_BILBIO, MATRICULE de TELEPHONE, MATRICULE de EMPRUNTEUR et DATE_DEBUT de EMP_CLOTURE ont été ignorées.

Pour la recherche des clés étrangères beaucoup de bruit a été généré comme par exemple NUM_OUVRAGE de ECRIT vers NUM_OUVRAGE de EMPRUNT ou TITRE de OUVRAGE vers TITRE de PROJET. Ce bruit pourra être élagué par l'analyse des données et des programmes. Seule l'auto jointure n'a pas été élicitée.

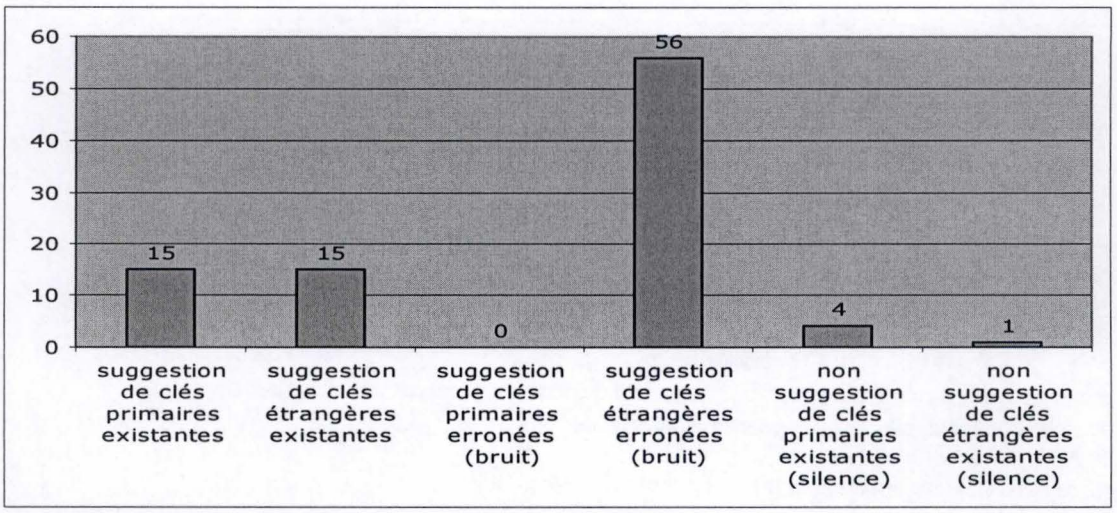


Figure 4-12 Quantification du résultat de la deuxième illustration.

4.4.3 Troisième illustration

La troisième illustration a été réalisée sur un cas concret à savoir le cas « WebCampus ». Webcampus[38] est un carrefour d'informations et de services associés aux cours dispensés aux Facultés Universitaires Notre Dame de la Paix à Namur.

Cette plate-forme est écrite en PHP et utilise une base de données MySQL.
La base de données utilisée par cette application est composée de deux schémas : un schéma principal comprenant 33 tables de 198 colonnes représentant les cours en ligne, les départements universitaires, les cours des utilisateurs, etc. et un schéma représentant les cours composé d'environs 50 tables par cours.

Dans cette illustration nous avons analysé le schéma principal. Lequel inclut 44 clés primaires explicites et 35 clés étrangères implicites[39] (figure 4-13).

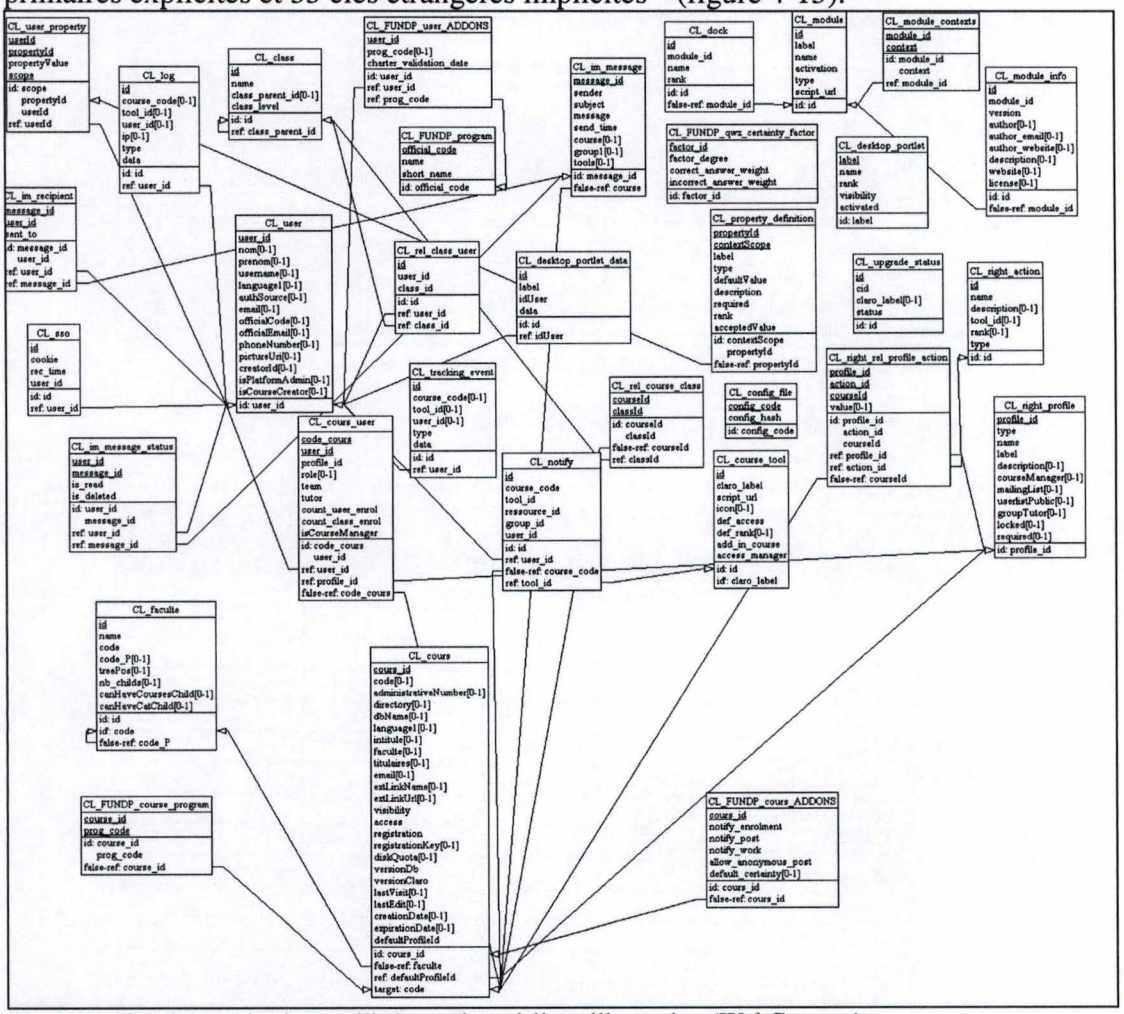
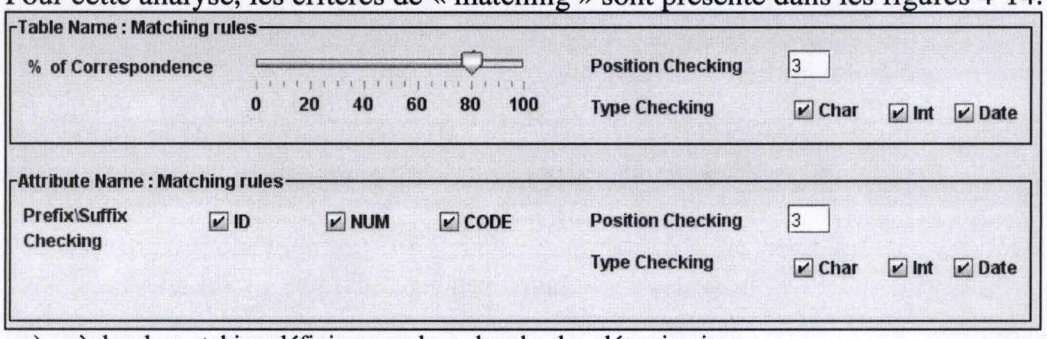


Figure 4-13 Schéma physique utilisé pour la troisième illustration (WebCampus).

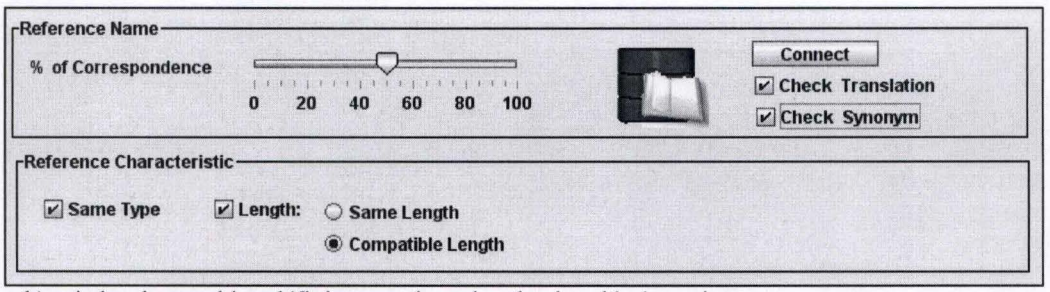
[38] De plus amples informations sont disponibles sur http://webcampus.fundp.ac.be
[39] Dans le code LDD de la base de données aucune clé étrangère n'est déclarée explicitement.

Le but de cette expérience est dévaluer l'efficacité de l'analyseur de schéma et de certaines de ces heuristiques. Plus précisément nous avons voulu voir si il était possible de découvrir certains indices permettant la détection des clés primaires et des clés étrangères enfuies dans le schéma.

Pour cette analyse, les critères de « matching » sont présenté dans les figures 4-14.



a) règles de matching définies pour la recherche des clés primaires.



b) règles de matching définies pour la recherche des clés étrangères.

Figures 4-14 Règles de matching utilisés pour la troisième illustration.

La figure 4-15 nous montre les heuristiques ayant été utiles pour la détection des indices.

	Heuristique	Heuristique Vérifiée
Clé primaire	Correspondance du nom de l'attribut avec entité	
	Préfixe/Suffixe ID	✓
	Préfixe/Suffixe NUM	
	Préfixe/Suffixe CODE	✓
	Position	✓
	Type de l'attribut (INT)	✓
	Type de l'attribut (CHAR)	✓
	Type de l'attribut (DATE)	
Clé étrangère	Correspondance des noms	✓
	Traduction	
	Synonyme	
	Type identique	✓
	Taille identique	✓
	Taille compatible	✓

Figure 4-15 Heuristiques vérifiées pour la troisième illustration.

Le schéma quantifiant les résultats est présenté dans la figure 4-16.

Pour la recherche des clés primaires, les clés : scope de CL_user_property, context de CL_module_contexts, label de CL_desktop_portlet et contextScope de CL_property_definition ont été ignorées par nos heuristiques. Un peu de bruit a cependant été produit. Celui-ci est dû à quelques attributs présentant un préfixe ou suffixe « ID » ou « CODE » et n'étant pas défini comme identifiant.

Comme la précédente illustration, la recherche des clés étrangères a généré beaucoup de bruit comme par exemple user_id de CL_cours_user vers user_id de CL_im_recipient ou course de CL_im_message vers courseId de CL_rel_course_class.
Notre analyse a également généré du silence. Sur les 17 clés étrangères non élicitées par notre analyseur, 9 clés présentaient des caractéristiques étranges. En effet, leur cible n'était pas déclarée comme identifiant. Leur découverte est donc rendue difficile par une simple analyse de schéma.
Pour les 8 autres clés non élicitées, nous pensons, après réflexion, qu'avec quelques petites améliorations[40] futures, celles-ci pourraient être retrouvées.

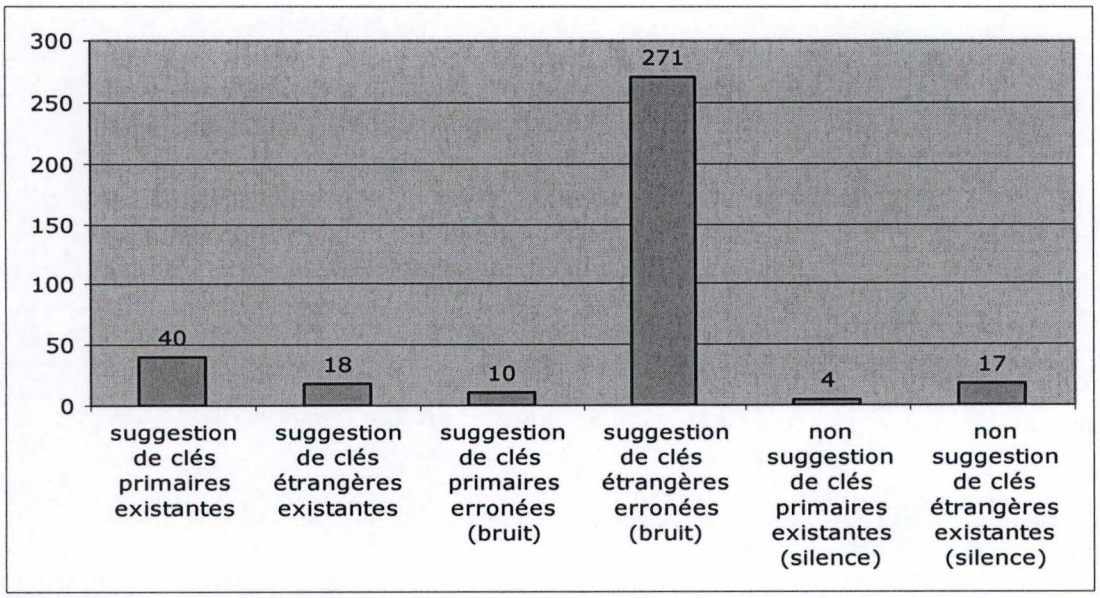


Figure 4-16 Quantification du résultat de la troisième illustration.

[40] Détection des auto-jointures et concaténation du nom de l'entité avec celui de son attribut pour augmenter la correspondance entre les clés étrangères cibles et origines. Pour ce dernier point nous pouvons citer en exemple classid de CL_rel_course_class vers id de CL_class.

4.5 Conclusion

Les quelques illustrations présentées dans ce chapitre ont montré que l'élicitation des clés primaires et étrangères à partir du schéma n'était pas une tâche aisée.

Les noms des relations et des attributs du schéma relationnel sont souvent abrégés ou ambigus. D'autres caractéristiques étranges peuvent également être présentes dans le schéma. Le type ou la taille ne matchent pas toujours entre un attribut cible et son origine; de même la cible d'une référence n'est pas toujours déclarée comme un identifiant.

L'analyseur de schéma semble être construit sur de bonnes bases. Cependant les limitations rencontrées sur l'étude d'un cas réel se sont révélées enrichissantes.
En effet, celles-ci ont permis de se faire une idée plus précise sur l'efficacité ou non de certaines de nos heuristiques et ont de ce fait ouvert la porte à de futures améliorations.
En effet, pour le matching des clés primaires la correspondance du nom de l'attribut avec celui de l'entité semble ne pas être fructueuse, mais pour le matching des clés étrangères il serait intéressant de combiner le nom d'une entité avec celui de son attribut pour augmenter la correspondance entre les clés étrangères cibles et origines.

L'analyse de schéma utilisée seule présente donc certaines limites. Il serait utopique de penser retrouver toute la sémantique simplement en regardant le schéma physique.

De plus, il existe une grande différence sémantique entre la richesse du schéma E/A[41] et la sobriété du schéma physique relationnel. La conversion du schéma conceptuel vers le schéma physique correspond à toute une série de transformations qui induisent une dégradation progressive du schéma conceptuel. Celui-ci devient donc moins complet, moins lisible et moins expressif.

Interroger les données et par la suite les programmes[42] semble donc inévitable pour obtenir une meilleure compréhension de la sémantique du système.

Combiner au schéma ces autres analyses permettra de réduire le bruit et le silence.

[41] Entité/Association.
[42] Certains indices peuvent être validés aisément en exploitant par exemple les informations sur les jointures présentes dans les programmes applicatifs.

Chapitre 5

ANALYSE DES DONNEES

5.1 Introduction

Le deuxième processus est une approche majeure de la rétro-ingénierie des bases de données. Il explore et analyse les données contenues dans la base de données.

Les données sont le résultat de l'exécution des mises à jour sur la base de données.
A travers cette analyse de contenu, certaines propriétés peuvent donc être élicitées et les hypothèses des contraintes implicites potentielles issues de l'étape précédente peuvent être renforcées, validées ou réfutées.

En effet, des résultats concluants ne peuvent être obtenus seulement en analysant l'information du schéma. Le scannage des données est un processus plus que nécessaire.

De plus, la distance sémantique entre les spécifications conceptuelles et l'implémentation physique est souvent plus courte via les données que via le code procédural. C'est pour ces raisons que les données suscitent un grand intérêt dans le domaine de la rétro-ingénierie.

Dans le monde, les données sont devenues fondamentales compte tenu du rôle vital et du volume qu'elles représentent pour le business de l'entreprise. Elles sont utilisées dans divers domaines tel que le marketing, les services bancaires, l'industrie, etc.

Elles suivent l'évolution de l'entreprise depuis ses débuts, c'est la source d'information la plus ancienne. On ne pourrait s'en passer : les programmes peuvent être réécrits, les interfaces peuvent être redéveloppés mais les données ne peuvent être ré-encodées, d'où l'importance de celles-ci dans le processus de rétro-ingénierie.

Par expérience, nous pouvons donc affirmer que les contraintes ne se retrouvent jamais dans une seule source d'information. Par exemple, certaines contraintes ne sont jamais validées dans le code procédural parce que celles-ci sont vérifiées par d'autres propriétés environnementales (fichier d'entrée toujours correct, écran validant les saisies, etc.).
Du coup, le seul moyen de valider une contrainte identifiée dans le schéma est de passer par l'analyse d'un autre composant tel que les données.

Leur utilité dans le domaine de la rétro-ingénierie des bases de données est multiple et variée. Le résultat de leur analyse, outre la validation d'hypothèses, peut aider à la découverte de relations « ISA» ([Lammari 1999], [Lammari 2007]).

5.2 Approche

5.2.1 Principes généraux

Notre approche se consacre à la validation d'hypothèses. Elle consiste à générer automatiquement des requêtes SQL de contrôle sur les données à partir des règles implicites élicitées par notre analyseur de schéma afin de vérifier que les données respectent les hypothèses émises.

Pour la génération du code SQL de ces requêtes, nous avons choisi un type de codage simple correspondant approximativement au standard SQL92[43] [Sql 1992].

Le but de cette analyse des données et de ces requêtes est de servir à la recherche et à la détection de propriétés pouvant mettre en évidence l'existence ou non existence de ces contraintes implicites potentielles. Celle-ci contribue donc à la vérification des hypothèses en y apportant des informations supplémentaires.

La difficulté de ce processus est de faire face à la grande quantité de données à analyser. En effet, celles-ci deviennent trop volumineuses pour être comprises et digérées facilement, surtout que dans le domaine informatique, bien plus qu'ailleurs le volume des données croit et évolue de jour en jour. De plus, il faut également faire attention au fait qu'une hypothèse validée par les données un jour n'est peut-être plus vraie un autre jour.

5.2.2 Automatisation

L'analyse de millions d'enregistrements est un travail répondant au besoin de l'automatisation car l'ingénieur n'aurait pas le temps et les capacités de réaliser cette lourde tâche manuellement. Un rapport reprenant les résultats de notre analyseur de données sera donc produit à cet effet.

Un échantillon peut également être constitué par l'ingénieur afin de limiter le temps de traitement de ce processus

5.2.3 Heuristiques

Les heuristiques de l'analyseur de données sont utilisées pour tenter de valider les hypothèses établies.

Nous classerons ces heuristiques en fonction de leur rôle.

[43] La norme SQL/92 a désormais pour nom SQL 2.

➢ *5.2.3.1 La découverte des clés primaires.*

Afin de vérifier la présence d'une ou plusieurs clés primaires dans la table, le générateur de requêtes doit s'assurer que toutes les instances de la clé identifiées sont uniques. Dans la figure 5-1, la contrainte d'unicité est vérifiée par les requêtes de la figure 5-3. Celles-ci sont générées par notre plugin. Elles comptent, dans un premier temps, le nombre d'enregistrements contenus dans la table, pour ensuite, compter le nombre d'enregistrements sans doublons contenus dans la table à partir d'une ou des clés primaires potentielles identifiées. La contrainte d'existence est vérifiée par la requête de la figure 5-2.

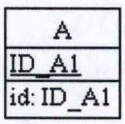


Figure 5-1 Schéma élémentaire abstrait incluant une clé primaire (ID_A1).

```
SELECT count(*) FROM A where ID_A1 is not null;
```

Figure 5-2 Requête abstraite vérifiant la contrainte d'existence de ID_A1 de la figure 5-1.

```
SELECT count(*) FROM A;
SELECT count(distinct ID_A1) FROM A;
```

Figure 5-3 Requêtes abstraites vérifiant la contrainte d'unicité de ID_A1 de la figure 5-1.

La figure 5-4 montre l'interprétation du résultat obtenu par l'exécution des trois requêtes.

Résultats des requêtes	Interprétation
Identiques	ID_A1 semble être une clé primaire de la table A.
Différents	ID_A1 ne semble pas être une clé primaire de la table A.

Figure 5-4 Interprétation des résultats des requêtes des figures 5-2 et 5-3.

➢ *5.2.3.2 La découverte des clés étrangères*

Afin de vérifier la présence d'une contrainte potentielle sur une clé étrangère, le générateur de requêtes doit s'assurer que toutes les instances de la table cible (enfant) sont référencées par au moins une instance de la table source (parent) afin de respecter la contrainte référentielle. Dans la figure 5-5 cette contrainte référentielle est vérifiée par la requête de la figure 5-6. Celle-ci est générée par notre plugin. Elle compte le nombre de valeurs référencées dans la table enfant non présentes dans la table parent.

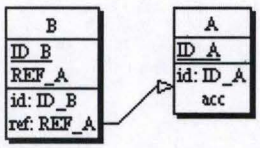


Figure 5-5 Schéma élémentaire abstrait incluant une clé étrangère.

```
SELECT count(*) FROM B where REF_A not in (select ID_A from A);
```

Figure 5-6 Requête abstraite vérifiant l'existence de la contrainte référentielle de la figure 5-1.

La figure 5-7 montre l'interprétation possible des résultats de la requête de la figure 5-6. Ces différentes interprétations sont issues des ouvrages de [Hainaut 2002] et de [Henrard 2003].

Résultat	Interprétation
n = 0	1. REF_A est une clé étrangère. 2. Coïncidence statistique; il se peut que dans le futur le résultat soit différent. REF_A n'est pas une clé étrangère.
0 < n < ε	1. REF_A n'est pas une clé étrangère. 2. REF_A est une clé étrangère, mais la requête a détecté des données erronées. 3. REF_A est une clé étrangère conditionnelle[44].
0 << n	1. REF_A n'est pas une clé étrangère. 2. REF_A est une clé étrangère conditionnelle.

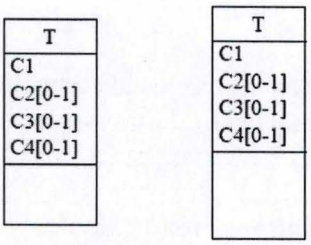
Figure 5-7 Interprétation des résultats de la requête de la figure 5-6.

L'intervention de l'ingénieur est nécessaire pour traiter les instances problématiques et pour décider de la suite à donner aux hypothèses sur base de l'interprétation des résultats

[44] Une clé étrangère conditionnelle est une clé étrangère pas toujours vérifiée. Celle-ci est seulement vérifiée quand certaines conditions sont satisfaites.

➢ *5.2.3.3 La découverte des contraintes de coexistence, d'exclusion et d'exactement-un.*

Nous ne parlerons que brièvement de ces contraintes, car l'analyse de celles-ci sort du périmètre de ce mémoire. La figure 5-8 illustre comment celles-ci peuvent être détectées par l'analyse des données. Si la requête donne un résultat, la contrainte est violée. Cet exemple a été repris de la thèse de [Hick 2001].




```
-- Contrainte de coexistence
SELECT * FROM t WHERE (c3 IS NULL OR c4 IS NULL) AND
                      (c3 IS NOT NULL OR c4 IS NOT NULL);
-- Contrainte d'exclusion
SELECT * FROM t WHERE (c2 IS NOT NULL OR c3 IS NOT NULL) AND c4 IS NOT NULL;
-- Contrainte exactement-un
SELECT * FROM t WHERE c4 IS NOT NULL;
```

Figure 5-8 Requêtes SQL permettant d'identifier les contraintes de coexistence, d'exclusion et d'exactement-un.

5.2.4 Approche finale et résultats

Les résultats de ces contrôles seront mis à la disposition de l'ingénieur afin de l'aider à infirmer ou confirmer ces règles implicites mais ceux-ci reste délicat à interpréter.

En effet, les données peuvent être erronées et il faut donc tenir compte d'un certain degré de contingence. C'est l'ingénieur qui se chargera de valider et de définir un pourcentage acceptable afin de valider ou de réfuter une hypothèse. Par exemple, si la contrainte est vérifiée par 85-90% des données, il est probable que cette contrainte existe, alors que si elle n'est respectée que par 10% des données, il faut la remettre en cause.
La qualité des résultats obtenus dépendra de l'exhaustivité de l'ensemble des données.

Par ailleurs, en plus de la validation des hypothèses, notre analyseur pourra servir de base pour évaluer et améliorer la qualité des données. Dans le chapitre 8, nous montrerons comment détecter les éventuelles données inconsistantes. Celles-ci pourront alors être soumises aux ingénieurs pour des corrections futures. Le processus de détection de ces données « erronées » est similaire au processus de validation des hypothèses. En effet, déterminer quelle proportion de données respecte une contrainte ou chercher les données qui violent cette contrainte est plus au moins analogue.

A ce stade, le résultat obtenu est un peu plus « complet » et « valide » grâce aux résultats des contrôles sur les données.

Nous présenterons dans la section suivante un exemple représentatif et clair illustrant l'importance de cette technique d'analyse.

5.3 Illustration

Dans notre illustration, nous supposerons que les données sont correctes et respectent les contraintes. Nous considérerons donc que si le résultat obtenu est 0, il existe une clé étrangère potentielle, a contrario, nous estimerons que celle-ci n'existe pas.

Pour illustrer le mécanisme de l'analyse des données, nous allons repartir de l'exemple simplifié de la « bibliothèque ». Les spécifications de la figure 5-9 représentent une bibliothèque dans laquelle les ouvrages (table OUVRAGE) sont écrits (table ECRIT) par aucun, un ou plusieurs auteurs (table AUTEUR).

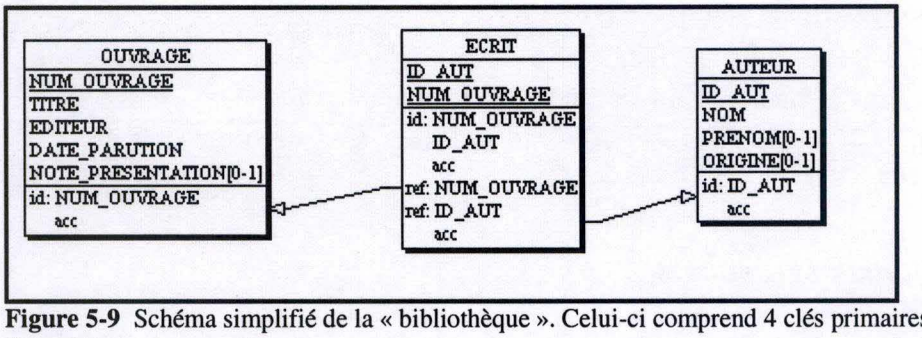


Figure 5-9 Schéma simplifié de la « bibliothèque ». Celui-ci comprend 4 clés primaires et 2 clés étrangères.

Les figures 5-10 et 5-11 illustrent les résultats obtenus et les requêtes générées suite à l'analyse du schéma de la figure 5-9. Les clés primaires et les clés étrangères obtenues ont été élicitées par l'analyseur de schéma illustré dans le chapitre 4.

La figure 5-12 montre un rapport généré automatiquement par notre analyseur suite à l'exécution des requêtes sur ces trois tables.

Clé primaire ?	Table	Requêtes générées	Interprétation
NUM_OUVRAGE	OUVRAGE	1) SELECT count(*) FROM OUVRAGE; 2) SELECT count(distinct NUM_OUVRAGE) FROM OUVRAGE; 3) SELECT count(*) FROM OUVRAGE where NUM_OUVRAGE is not null;	Le résultat des trois requêtes est identique. La clé NUM_OUVRAGE semble constituer un identifiant potentiel de OUVRAGE.
ID_AUT	ECRIT	1) SELECT count(*) FROM ECRIT; 2) SELECT count(distinct ID_AUT) FROM ECRIT; 3) SELECT count(*) FROM ECRIT where ID_AUT is not null;	Le résultat des trois requêtes est différent. La clé ID_AUT semble ne pas présenter les caractéristiques pour constituer un identifiant potentiel de ECRIT.
NUM_OUVRAGE	ECRIT	1) SELECT count(*) FROM ECRIT; 2) SELECT count(distinct NUM_OUVRAGE) FROM ECRIT; 3) SELECT count(*) FROM ECRIT where NUM_OUVRAGE is not null;	Le résultat des trois requêtes est différent. La clé NUM_OUVRAGE semble ne pas présenter les caractéristiques pour constituer un identifiant potentiel de ECRIT.
ID_AUT NUM_OUVRAGE	ECRIT	1) SELECT count(*) FROM ECRIT; 2) SELECT count(distinct ID_AUT,NUM_OUVRAGE) FROM ECRIT; 3) SELECT count(*) FROM ECRIT where ID_AUT is not null and NUM_OUVRAGE is not null;	Le résultat des trois requêtes est identique. Les clés ID_AUT, NUM_OUVRAGE semblent constituer un identifiant potentiel de ECRIT.
ID_AUT	AUTEUR	1) SELECT count(*) FROM AUTEUR; 2) SELECT count(distinct ID_AUT) FROM AUTEUR; 3) SELECT count(*) FROM AUTEUR where ID_AUT is not null;	Le résultat des trois requêtes est identique. La clé ID_AUT semble constituer un identifiant potentiel de AUTEUR.

Figure 5-10 Analyse par les données des clés primaires de la figure 5-9.

Clé étrangère ?	Requêtes générées	Interprétation
de ECRIT.NUM_OUVRAGE vers OUVRAGE.NUM_OUVRAGE	SELECT count(*) FROM ECRIT where NUM_OUVRAGE not in (select NUM_OUVRAGE from OUVRAGE);	n=0 : NUM_OUVRAGE de ECRIT peut constituer une clé étrangère potentielle vers NUM_OUVRAGE de OUVRAGE.
de ECRIT.ID_AUT vers AUTEUR.ID_AUT	SELECT count(*) FROM ECRIT where ID_AUT not in (select ID_AUT from AUTEUR);	n=0 : ID_AUT de ECRIT peut constituer une clé étrangère potentielle vers ID_AUT de AUTEUR.
de OUVRAGE.NUM_OUVRAGE vers ECRIT.NUM_OUVRAGE	SELECT count(*) FROM OUVRAGE where NUM_OUVRAGE not in (select NUM_OUVRAGE from ECRIT);	n<>0 : NUM_OUVRAGE de ECRIT semble ne pas être une clé étrangère potentielle vers NUM_OUVRAGE de ECRIT.
de AUTEUR.ID_AUT vers ECRIT.ID_AUT	SELECT count(*) FROM AUTEUR where ID_AUT not in (select ID_AUT from ECRIT);	n<>0 : ID_AUT de AUTEUR semble ne pas être une clé étrangère potentielle vers ID_AUT de ECRIT.

Figure 5-11 Analyse par les données des clés étrangères de la figure 5-9.

DBRE - Data Analysis - Result Primary Key for : BIBLIO_SIMPLE									
N°	Attribute Name	Table Name	Total Records	% Null	% Not Unique	Threshold %Null - %Not Unique	In data ?	In pattern ?	Primary key ?
1	ID_AUT	AUTEUR	40	0%	0%	0% - 0%	yes	yes/no	yes/no
2	ID_AUT	ECRIT	40	0%	50%	0% - 0%	no	yes/no	yes/no
3	NUM_OUVRAGE	ECRIT	40	0%	50%	0% - 0%	no	yes/no	yes/no
4	ID_AUT,NUM_OUVRAGE	ECRIT	40	0%	0%	0% - 0%	yes	yes/no	yes/no
5	NUM_OUVRAGE	OUVRAGE	40	0%	0%	0% - 0%	yes	yes/no	yes/no

Figure 5-12 Exemple du rapport généré par notre analyseur de données.

NB : Cet exemple a été testé dans l'environnement MYSQL[45]. A cet effet les trois tables de l'exemple ont été crées et alimentées avec quelques données fictives respectant les contraintes illustrées.

5.4 Conclusion

L'analyse des données peut aider à lever le voile sur les hypothèses émises par l'analyse du schéma. Ce processus contribue à l'élicictation des contraintes implicites.

Cependant, cela suppose que pour retrouver ces contraintes dans les données, des règles de gestion doivent être implémentées d'une manière ou d'une autre. Cela nous laisse donc supposer que beaucoup de contraintes sont gérées soit dans le code LDD de la base de données ou soit de manière procédurale (programmes, trigger, procédures, etc.).

[45] MySQL signifie « My Structured Query Language » en anglais. C'est un SGBDR Open Source.

Chapitre 6

ANALYSE DES PATTERNS

6.1 Introduction

Nous voilà donc dans le chapitre consacré à l'analyse de code par détection de patterns.

Comme énoncé dans l'état de l'art, beaucoup d'auteurs estiment que pour comprendre la signification des données, il faut partir du code applicatif. Ils reconnaissent également que le code procédural d'une application est une source d'information importante pour la rétro-ingénierie des bases de données. En effet, les contraintes et structures de données qui ne sont pas déclarées explicitement lors de la déclaration de la base de données s'y trouvent codées d'une manière ou d'une autre.

Les programmes sont la source d'information la plus à jour du système et le langage de programmation est très précis et déterministe. Les informations acquises lors de cette analyse peuvent donc être considérées comme fiables dans la plupart des cas.

Dans le monde informatique, une grande part de la gestion des données est enfuie dans le code. Ce processus tentera de comprendre comment ces données sont stockées et manipulées dans le code procédural. Dans le but de limiter cette tâche fastidieuse et difficile de recherche, notons que lorsque nous parlons de code procédural, c'est le plus souvent pour en considérer les requêtes SQL afin d'y rechercher les « patterns » éventuels.

En effet, les anciens systèmes de gestion de base de données n'offrent pas la possibilité de définir certaines contraintes complexes. Il convient donc au programmeur d'implémenter ces contraintes dans le code applicatif.
Le programme doit donc valider toutes ces contraintes avant de mettre à jour les données. De même lorsqu'un programme accède aux données pour générer un rapport, celui-ci fait un usage direct de ces contraintes.

Leur analyse est plus compliquée que pour le schéma et les données. Bien souvent, les rétro-concepteurs ont besoin d'outils et de techniques de compréhension de programmes, ils doivent disposer d'un parseur, d'un « pattern miner », des sources disponibles, etc.

De plus, les parties du code implémentant ces contraintes sont généralement disséminées dans un grand nombre de programmes. Chaque programmeur à son propre style de programmation et bien souvent les mêmes contraintes se retrouvent codées de manières différentes, une même contrainte pouvant être dupliquée à plusieurs endroits.

Les patterns sont un des moyens les plus simples pour rechercher de l'information dans le code de programmes sources.

Les programmeurs bien disciplinés utilisent des patterns standards pour résoudre des problèmes communs. Par conséquent, un pattern donne une réponse à un problème qui a été identifié. Rechercher une (des) instance(s) de ces patterns dans le code source nous permet de repérer où ce genre de problème a été résolu ([Signore 1994], [Petit 1994], [Henrard 2003]).

Dans ce chapitre nous devons tout d'abord définir ce que nous entendons par le terme « patterns », pour ensuite définir le scope de notre approche. Celui-ci sera principalement de définir un catalogue de patterns afin de mettre la main sur certaines contraintes implicites enfuies dans le code.

6.2 Les patterns

L'utilisation du terme « pattern », traduisible au mieux en motif ou cliché, a été introduite par l'architecte anglais Christopher Alexander dans les années 70 ([Alexander 1964] [Alexander 1977], [Alexander 1979]).

Un pattern est donc souvent utilisé pour désigner un modèle ou une structure.
Dans notre usage, celui-ci peut-être vu comme un ensemble de chaînes de caractères pouvant contenir des variables, des fonctions, des « wildcards »[46], du code et même d'autres patterns. C'est donc un fragment de code procédural comportant des instructions de programmation entouré d'un ou plusieurs ordres SQL.
Ces clichés reposent principalement sur l'enchaînement d'opérations de manipulation de données relationnelles. Leur usage peut faire penser à l'existence de contraintes implicites.

Pour interroger ou gérer la base de données relationnelles, ces fragments ou instructions d'accès et de manipulation de données s'appuient sur les requêtes SQL et sur les spécifications LMD[47].

On peut considérer le pattern comme un flux de données entre les programmes applicatifs et la base de données et le définir comme un enchaînement naturel d'opérations utilisé pour consulter et assurer la bonne gestion de la base de données.

Leur composition donne une indication sur les contraintes implicites se trouvant dans le code et veillent à la cohérence des données de l'application. En effet, la plupart de ceux-ci s'occupent de la gestion des contraintes.

---

[46] Dans le langage SQL, ils se caractérisent par les caractères « % », « _ », « ? », « * ».
[47] Langage de manipulation de données - DML en anglais, il permet de sélectionner, d'insérer, de modifier ou de supprimer des données dans une table d'une base de données relationnelle.

De plus, les requêtes sont écrites dans un langage structuré de requêtes (SQL[48]); langage obéissant à une syntaxe formelle stricte manipulant des types de données simples. L'expression de contraintes dans ce langage se retrouve souvent codée avec les mêmes caractéristiques, avec une structure généralement standard et normalisée pouvant être facilement reconnaissable dans le langage hôte; ce qui nous facilitera la tâche dans la création du catalogue de patterns.

Le but de notre recherche est donc de trouver une certaine standardisation dans le codage des contraintes. Cette analyse de patterns est donc applicable à des systèmes développés à l'aide de langages standards de troisième génération tels que COBOL/SQL ou C/SQL.

Dans ce contexte, ceux-ci vont nous servir avant tout à repérer la solution implémentée pour résoudre un problème de gestion de contraintes n'ayant pas été défini explicitement dans le langage de définition de données (LDD) de la base de données.

6.3 Approche et limitation

6.3.1 Principes généraux

L'approche illustrée ici entre dans le domaine des techniques de compréhension de programmes. En raison de la difficulté et du temps imparti, nous nous limiterons à l'élaboration d'un catalogue reprenant les clichés les plus classiques dit « naturels » et non à la recherche automatique de ces patterns.

Ces clichés enfuis dans le langage hôte ne sont pas à négliger. Ces patterns SQL sont les seuls points d'entrée dont dispose le programmeur pour accéder aux données de la base.

Notre idée est donc d'effectuer une analyse syntaxique des requêtes représentatives de l'activité de la base de données dans le but de comprendre la navigation logique entres les différents objets, pour ensuite réaliser l'extraction de la sémantique à partir de cette navigation. Cette étape de compréhension peut, non seulement nous permettre de valider des hypothèses définies lors des étapes précédentes, mais également permettre la découverte de nouvelles contraintes.

Il semble plus raisonnable à notre niveau de limiter notre approche à une inspection visuelle de ces requêtes dites « statiques » et de commencer une recherche manuelle pour extraire les fragments de code SQL se trouvant dans les programmes applicatifs.

L'analyse se fera en deux phases. Une phase d'extraction et de repérage des modules à analyser, suivie d'une phase d'inspection visuelle de ceux-ci afin d'identifier les patterns « matchant » à notre catalogue.

[48] Structured query language en anglais.

Nous verrons à travers des exemples comment certaines contraintes, n'ayant pas pu être définies explicitement dans le langage de définition de données, se retrouvent noyées dans ces patterns.

La principale difficulté dans cette analyse réside dans la variété de représentation syntaxique d'une requête (forte variabilité). Par exemple, une jointure peut apparaître sous différentes formes (jointures, utilisation de boucles FECTH, etc.).

C'est la phase de notre processus où l'intervention de l'ingénieur est la plus sollicitée.

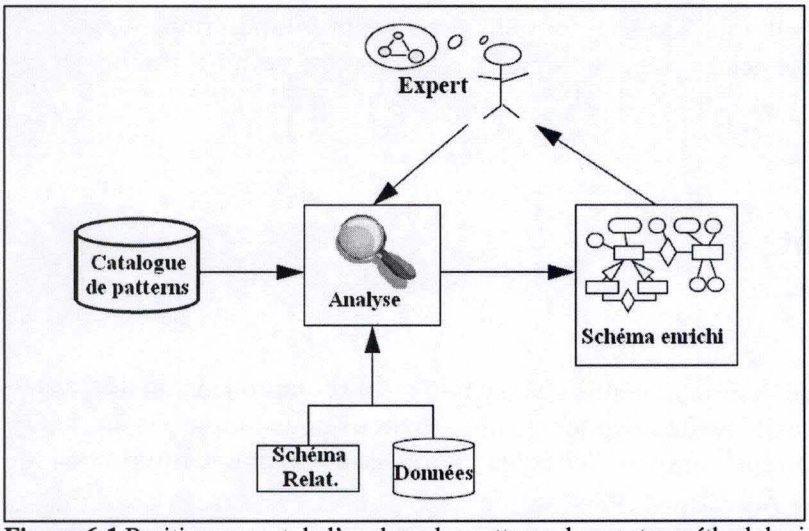


Figure 6-1 Positionnement de l'analyse des patterns dans notre méthodologie.

6.3.2 Difficulté de l'approche

Les programmes sont souvent modifiés et ne sont donc pas très stables.

Même si c'est la source d'information la plus fiable, cela ne veut pas dire qu'il ne peut y avoir des erreurs dans les programmes. En effet le programmeur peut très bien avoir oublié de valider une contrainte dans un fragment de code ou alors il peut s'agir d'une contrainte optionnelle ou de code mort.

L'analyse de programmes est complexe et fastidieuse, les applications du monde de l'entreprise englobent une grande quantité de programmes et gèrent des milliers de lignes de codes, ce qui a pour conséquence de venir surcharger le processus d'analyse. L'utilisation du code source est donc complexe, un programme n'est pas simplement une juxtaposition de patterns. Ceux-ci se retrouvent bien souvent dispersés dans le code source.

De plus, les résultats de l'analyse dépendent essentiellement de paramètres techniques mais également de paramètres non techniques comme le style de programmation du

développeur. Les programmeurs peuvent être non disciplinés et utiliser des techniques diverses et non standards pour résoudre des problèmes, et ce pour différentes raisons (performance, manque de connaissances, etc.).
Par exemple, nous pouvons définir une contrainte référentielle de manière explicite dans le code SQL-DDL de la manière suivante « ...foreign key <colonne> reference <table> ». Cette déclaration est simple à détecter, l'ingénieur est sûr de son existence, il peut donc l'intégrer dans le schéma sans aucune autre vérification.
A l'inverse, il existe différentes façons de coder une même contrainte référentielle dans les programmes applicatifs, ce codage dépend bien souvent de l'humeur, de l'expérience et des connaissances du programmeur. Cette variation syntaxique augmente un peu plus la difficulté de l'approche.

6.3.3 Heuristiques

Notre domaine de recherche porte sur la détection des clés étrangères (contraintes référentielles) et des clés primaires.

Pour la recherche des contraintes référentielles implicites, il existe deux situations de détection de patterns potentiels.

La première situation se focalise sur une phase dite de validation. En effet, s'il existe une contrainte de référence, les données doivent d'abord être validées avant d'être enregistrées dans la table. Avant de mettre à jour ou de supprimer une valeur dans une table (insert, update, delete) et afin de garantir le respect de cette contrainte, il faut vérifier l'existence d'un fragment de code (pattern) qui contrôle et valide la valeur de(s) attribut(s) de la table d'origine vers la table de référence. Généralement cette validation se fait via une requête de sélection (select). La présence d'un pattern de validation permet d'éliciter une contrainte référentielle potentielle.

La phase de recherche d'information constitue la seconde situation pouvant signaler la présence d'une contrainte référentielle potentielle.
En effet, pour accéder à la table, la valeur de l'attribut référentiel est utilisée pour lire la table référencée via la clé d'accès.
C'est lors de la phases de recherche d'information comme lors de la génération de rapports ou de recherche d'informations sur les données (select) qu'on se retrouve en présence d'un pattern de recherche permettant d'éliciter une contrainte référentielle potentielle.

Pour la détection des clés primaires, la lecture (select) ne s'opère pas sur une table d'origine et sur une ou plusieurs tables référencées mais se focalise sur la même entité. Notre but sera, en partant de cette lecture, de vérifier les contraintes d'unicité et d'existence de la clé primaire.

Ces heuristiques permettent de limiter l'espace de recherche afin de trouver et de définir plus facilement des patterns intéressants dans le but d'enrichir le contenu de notre catalogue.

6.4 *Le catalogue*

6.4.1 Introduction

Le catalogue peut être vu comme une bibliothèque répertoriant les patterns (clichés) du domaine relationnel.
Les quatre types de requêtes par lesquelles les programmes établissent des liens avec les données sont : la sélection (SELECT), l'insertion (INSERT), la suppression (DELETE) et la modification (UPDATE) d'instances d'une table.
Les requêtes contiennent, mis à part les mots-clés spécifiques à SQL, des noms de tables, des noms de colonnes et des noms de variables ou des curseurs dans lesquels des valeurs de colonnes sont stockées.

Le catalogue peut constituer un outil précieux pour nos analyses. En effet, la présence de ces différents clichés dans le code source, identifiés par un « matching » manuel, tentera avant tout d'éliciter ou non certaines contraintes implicites, mais également de confirmer ou d'infirmer certaines hypothèses issues de l'analyse du schéma et des données. Celui-ci peut donc constituer une source d'information précieuse pour la rétro-ingénierie.

Bien entendu, les hypothèses émises via les patterns peuvent-être validées par les données ou par le schéma, comme énoncé dans notre chapitre illustrant notre méthodologie.

Le catalogue contiendra une série de patterns génériques intéressants, dans lesquels nous trouverons des requêtes SQL « abstraites », où les noms de tables, de colonnes seront représentés par des variables.

De plus, notre catalogue, grâce à l'identification des patterns dit de « contrôle », peut jouer un rôle dans le volet qualité. Celui-ci sera d'épauler l'ingénieur afin de lui permettre de mettre l'accent sur les fragments de code applicatif d'apparence suspecte.

La tâche de l'ingénieur sera alors de vérifier manuellement qu'une requête de contrôle précède bien chaque pattern présentant des « opérations critiques », opérations pouvant potentiellement mettre en péril l'intégrité des données. En d'autres termes, l'opération de mise à jour ne sera exécutée, que si et seulement si, le résultat de la requête de contrôle est satisfaisant. Cette étape de validation constituera un pas important dans l'amélioration de la qualité.

Il existe une grande quantité de patterns intéressants illustrant l'usage ou la gestion de contraintes. Les cataloguer tous serait utopique.

6.4.2 Constitution du catalogue

Le contenu de notre catalogue a d'abord été enrichi par notre générateur de triggers (chapitre 8) et par notre analyseur de donnée (chapitre 5). En effet, ceux-ci font un usage direct de certains patterns.

Il a ensuite été complété par nos diverses lectures, par les travaux de [Signore 1994], [Petit 1994], [Hainaut 1997c] et [Hainaut 2002], par nos réflexions et notre expérience du terrain.

Nous tâcherons d'utiliser un vocabulaire commun pour décrire chacun de nos patterns; celui-ci sera basé sur la grammaire SQL [Dat 1989] et les instructions seront écrites en pseudo-code[49].

Les patterns seront classés en fonction de leurs rôles et de leurs caractéristiques syntaxiques.

6.4.3 Contenu du catalogue

Les patterns du catalogue sont introduits en fonction de leur rôle. Rappelons que ce catalogue est loin d'être exhaustif.

➢ *6.4.1.1 La découverte des clés primaires.*

a) Patterns génériques permettant de supposer la non présence d'une clé primaire.

N°	Pattern	Description
1	**Pas de requête de type :** SELECT DISTINCT <sélection> FROM A WHERE a_1 = <expression scalaire> | <var-hôte[50]> [AND…]	Le mot clé « DISTINCT » est utilisé en SQL pour éliminer les doublons. Son utilisation implique une valeur non unique. Celle-ci peut indiquer que les attributs se trouvant dans la clause WHERE ne font pas partie de la clé primaire. NB : Beaucoup de programmeurs font un usage erroné de ce mot clé, l'utilisant abusivement et contribuant ainsi à donner un faible poids à ce pattern.

[49] Le pseudo-code est une façon de décrire un algorithme sans référence à un langage de programmation en particulier.

[50] Le SQL imbriqué utilise des variables hôtes pour échanger des informations entre les instructions SQL et le code source. On peut accéder à n'importe quel champ d'une variable hôte en préfixant simplement le nom du champ par un signe deux-points.

N°	Pattern	Description
2	**Pas de déclaration de curseur :** DECLARE <curseur-id> FOR SELECT <sélection> FROM A WHERE a_1 = <expression scalaire> | <var-hôte> [AND…] **Suivi de** OPEN <curseur-id> **Et d'une boucle contenant :** FETCH <curseur-id> INTO <liste-de-var-hôte> **ou** sans affectation des lignes sélectionnées. **Suivi de** CLOSE <curseur-id>	Le curseur est utilisé lorsque la table résultante d'un ordre select peut comporter plusieurs lignes. Son utilisation peut indiquer que les attributs se trouvant dans la clause WHERE ne font pas partie de la clé primaire. En effet si les attributs identifiants la table étaient présents dans la clause WHERE, le select n'aurait pas besoin de curseur car celui-ci renverrait un et un seul résultat.

N°	Pattern	Description
3	**Pas de requête de type :** SELECT < sélection > [, Fct_Agreg (ChampFA)[51]] FROM A [WHERE a_1 = <expression scalaire> | <var-hôte> [AND…] GROUP BY <selection de regroupement> [HAVING <condition de regroupement >]	La clause GROUP BY permet de faire un regroupement sur un ensemble de lignes. Son utilisation peut indiquer que les attributs se trouvant dans la clause GROUP BY ne font pas partie de la clé primaire.

N°	Pattern	Description
4	**Pas de requête de type :** SELECT [<sélection>,] Fct_Agreg (ChampFA) FROM A WHERE a_1 = <expression scalaire> | <var-hôte> [AND…]	Les agrégats sont utilisés en SQL pour calculer. En exemple nous pouvons citer : le count (comptage), le sum (somme), l'avg (moyenne), le max (maximum) et le min (minimum). Leur utilisation indique que les attributs se trouvant dans la clause WHERE ne font pas partie de la clé primaire.

[51] Fonction d'agrégation : {AVG | MAX | MIN | SUM | COUNT}.

b) Patterns génériques permettant de supposer la présence d'une clé primaire.

N°	Pattern	Description
5	**Bloc d'instructions de type :** SELECT <sélection> [INTO <var-hôte>] FROM A WHERE a_1 = <expression scalaire> | <var-hôte> [AND...] **Suivi de :** SI (SQLCODE[52] <oper. de comparaison > <nb[53]> | <var-hôte> <oper. de comparaison > <nb>) ALORS INSERT INTO A ... | UPDATE A ... | DELETE FROM A ... SINON ERREUR FIN-SI	Ce pattern procédural est généralement utilisé lors d'opérations de mise à jour sur la clé primaire. En effet, chaque instruction de modification est précédée d'une requête de validation de la contrainte. Les attributs présents dans la clause WHERE de la requête de vérification (requête précédent les opérations de mise à jour) contient les clés pouvant faire partie de la clé primaire. Ce pattern vérifiera l'existence ou non existence de la clé primaire avant de procéder à la mise à jour de celle-ci.

6.4.1.2 La découverte des clés étrangères.

Patterns génériques permettant de supposer la présence de clés étrangères.

N°	Pattern	Description
6	**Requête de type :** SELECT <sélection > FROM A ,B ... WHERE ... $A.a_1 = B.b_2$ ...	Ce pattern suggère que l'attribut b_2 de B est une clé étrangère vers a_1 de A. Il suggère également qu'a_1 est une clé primaire de A. En effet, la présence de jointures peut indiquer la présence de clés étrangères entre les tables A et B.

[52] Variable de communication entre le programme et la base de données. Les principales valeurs du SQLCODE sont 0 pour succès, 100 pour "no data", -1 pour erreur. Celles-ci permettent de suivre le bon déroulement des opérations.
[53] Nombre entier.

N°	Pattern	Description
7	**Requête de type :** SELECT <sélection> FROM B WHERE b_2 [NOT] IN (SELECT a_1 FROM A WHERE ...)	La requête SQL imbriquée de ce pattern, permet de faire une jointure entre les tables A et B. Ce pattern suggère que l'attribut b_2 de B est une clé étrangère vers a_1 de A. Il peut indiquer également qu'a_1 est une clé primaire de A. Il est souvent utilisé pour vérifier l'existence de la contrainte d'intégrité référentielle.

N°	Pattern	Description
8	**Bloc d'instructions de type :** SELECT <sélection> INTO <output-var> FROM A [WHERE ... a_1 = <expression scalaire> | <var-hôte> ... [AND ...]] ... SELECT <sélection> FROM B WHERE ... b_2 = <input-var> ... [AND ...] **NB:** <output-var> ≡ <input-var>	La requête utilisée dans ce pattern est identique aux jointures définies dans les patterns 6 et 7. A la seule différence qu'elle est codée de manière procédurale L'utilisation des variables hôtes « output » et « input » permet d'établir le lien entre les deux requêtes (output-input dépendance). En effet, la variable hôte utilisée pour l'output de la première requête est la même que celle utilisée pour l'input de la seconde requête. Ce qui nous fait penser à l'usage d'une clé étrangère. Ce pattern suggère que l'attribut b_2 de B est une clé étrangère vers a_1 de A. Il peut indiquer également qu'a_1 est une clé primaire de A.

N°	Pattern	Description
9	**Bloc d'instructions de type :** **1) déclaration de curseur pour A :** DECLARE <curseur-id> FOR SELECT <sélection> FROM A [WHERE a_2 = <expression scalaire> | <var-hôte>... [AND ...]] **Suivi de :** OPEN <curseur-id> **Et d'une boucle contenant :** FETCH <curseur-id> INTO <output-var>, [<liste-de-var-hôte>] **Suivi de :** CLOSE <curseur-id> **NB:** Les attributs constituant la clé primaire de A ne doivent pas être présents dans la clause where car la boucle fetch n'aurait alors aucun sens. **2) déclaration de curseur pour B :** DECLARE <curseur-id> FOR SELECT <sélection> FROM B WHERE b_2 = <input-var>... [AND ...] **Suivi de :** OPEN <curseur-id> **Et d'une boucle contenant :** FETCH <curseur-id> INTO <output-var>, [<liste-de-var-hôte>] **Suivi de :** CLOSE <curseur-id> **NB:** - <output-var> ≡ <input-var> - Les attributs constituant la clé primaire de B ne doivent pas être présents dans la clause where car la boucle fetch n'aurait alors aucun sens.	Ce pattern est presque identique au pattern précédent. Celui-ci utilise deux boucles fetch imbriquées pour réaliser la jointure entre les table A et B. L'utilisation des variables hôtes « output » et « input » permet d'établir le lien entre les deux requêtes (output-input dépendance). En effet, la variable hôte utilisée pour l'output de la première requête est la même que celle utilisée pour l'input de la seconde requête. Ce qui nous fait penser à l'usage d'une clé étrangère. Ce pattern suggère que l'attribut b_2 de B est une clé étrangère vers a_1 de A. Il peut indiquer également qu'a_1 est une clé primaire de A.

N°	Pattern	Description
10	**Bloc d'instructions de type :** SELECT <sélection> [INTO <output-var>] FROM A WHERE a_1=<expression scalaire> | <var-hôte> ... [AND...] **Suivi de :** SI (SQLCODE <oper. de comparaison > <nb entier> | < output-var> <oper. de comparaison > <nr entier>) ALORS INSERT INTO B (..., b_2 , ...) VALUES (..., <input-var> | <var-hôte>, ...) | UPDATE B SET (b_2 = <input-var> | <var-hôte>, ...) WHERE b_1 = ... [AND ...] SINON ERREUR FIN-SI **NB:** <output-var> ≡ <input-var>	Pattern procédural pouvant assurer la contrainte d'intégrité référentielle lors d'une opération d'insertion (INSERT) ou de modification (UPDATE) sur la table de référence (table enfant). Chaque instruction de modification est précédée d'une requête de validation de la contrainte référentielle potentielle. Ce pattern suggère que l'attribut b_2 de B est une clé étrangère vers a_1 de A. Il peut indiquer également qu'a_1 est une clé primaire de A. La présence de l'output-input dépendance peut également faire penser à l'existence d'une clé étrangère.

N°	Pattern	Description
11	**Bloc d'instructions de type :** **Pattern n°5 en ne tenant compte que des instructions UPDATE et DELETE** ... **Suivi de :** **a) En cas d'UPDATE de la clé primaire :** UPDATE B SET (b_2 = ...) WHERE b_2 = ... **b) En cas de DELETE de la clé primaire :** UPDATE B SET (b_2 = <valeur par défaut> | null) WHERE b_2 = ... | DELETE FROM B WHERE b_2 = ...	Pattern procédural pouvant assurer la contrainte d'intégrité référentielle lors d'une opération de suppression (DELETE) ou de modification (UPDATE) de la clé primaire de la table parent (table d'origine). Lors d'une mise à jour de la clé primaire de la table parent, les modifications sur la table enfant peuvent être stoppées ou être réalisées en cascade. Lors de la suppression de la clé primaire de la table parent. Trois options sont offertes à l'ingénieur : - Suppression en cascade - Valeur par défaut ou nulle - Ne rien faire (erreur) Chaque instruction de modification est précédée d'une requête d'existence de la clé primaire (voir pattern n°5). Ce pattern suggère que l'attribut b_2 de B est une clé étrangère vers a_1 de A. Il peut également faire penser qu'a_1 est une clé primaire de A (via présence pattern n°5).

Pour alléger la tâche de l'analyse des codes sources, certains de nos patterns peuvent être identifiés dans les programmes via le « pattern-matching engine » de l'atelier DB-MAIN. Les patterns à rechercher sont alors définis dans un langage de définition de patterns (Pattern Description Language) fort proche de la notation BNF[54].

Les patterns comprenant des blocs d'instructions sont plus complexe à analyser. Ceux-ci requièrent des techniques d'analyse bien plus avancées qu'un simple matching. Pour détecter ce type de patterns nous devons utiliser des techniques complexes de compréhension de programmes ([Henrard 2002], [Henrard 2003]).

Dans le cadre de ce travail nous limiterons notre approche à une analyse visuelle de ces patterns.

6.5 Illustration

Dans le but d'illustrer, sur des exemples concrets, l'importance des patterns dans la rétro-ingénierie nous avons repris l'énoncé d'un travail exploratoire réalisé dans le cadre de ce mémoire.

L'énoncé de ce travail était le suivant : « *Considérons un schéma de base de données relationnelle dans lequel aucune clé étrangère n'apparaît explicitement.*
Dans ce schéma, on trouve (entre-autres) une table CLIENT (NCLI, NOM, LOCALITE) et une table COMMANDE (NCOM, CLI, DATECOM).
En discutant avec les programmeurs, on apprend que la valeur de CLI de toute commande doit toujours correspondre à la valeur de NCLI d'un client.
En d'autres termes, le schéma comprend une clé étrangère implicite de COMMANDE vers CLIENT. Cette contrainte n'étant pas connues du SGBD, ce sont les programmes d'application qui doivent la gérer à chaque mise à jour de la base de données ».

Pour enrichir la complexité de ce travail, nous avons également supposé que les clés primaires étaient elles aussi implicites. Le schéma de la figure 6-2 représente la description des deux tables définies dans l'énoncé.

Pour illustrer cet exemple, nous avons réalisé deux versions d'un petit programme COBOL/Embedded SQL dont le code source est disponible en annexe.
La première version fait une utilisation de la clause « WHERE » pour réaliser la jointure logique entre les deux tables et fait usage du SQLCODE pour récupérer le résultat des requêtes. La seconde version met en œuvre l'utilisation des boucles « fetch » pour réaliser les jointures et utilise une variable hôte pour tester le résultat des requêtes plutôt que le SQLCODE.

[54] La forme de Backus-Naur (souvent abrégée en BNF, de l'anglais Backus-Naur Form) est une notation permettant de décrire les règles syntaxiques des langages de programmation [Wikipédia].

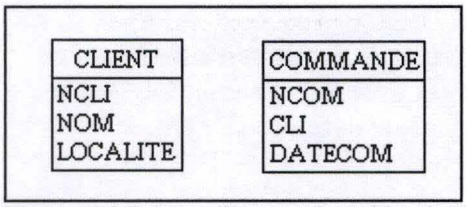


Figure 6-2 Schéma illustrant les tables de l'énoncé de gestion clients-commandes.

Les fonctionnalités de ce programme sont :

- Création d'une commande
- Création d'un client
- Changer la valeur de NCLI d'un client
- Changer la valeur de CLI d'une commande
- Suppression d'une commande
- Affichage des commandes passées dans une localité donnée

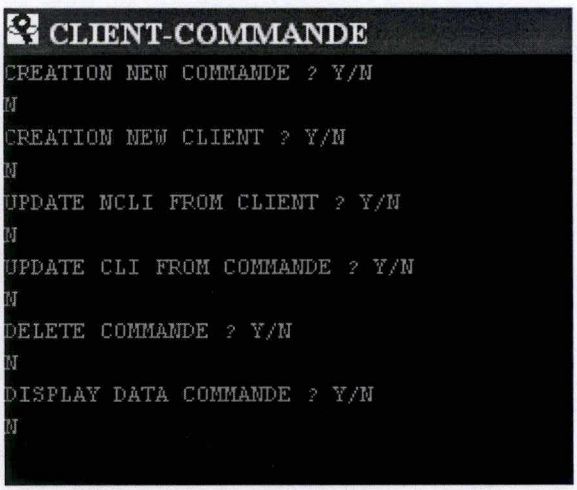


Figure 6-3 Interface graphique de notre application de gestion des clients et des commandes.

NB : Cet exemple a été testé à l'aide du compilateur PERCOBOL sur une base de donnée crée dans l'environnement MYSQL.

Voici quelques exemples de patterns identifiés dans le code source des deux versions du programme de gestion des clients et des commandes. Ceux-ci ont été classés en fonction des fonctionnalités de l'application. Toutes les fonctionnalités ne seront pas évoquées.

a) Création d'une commande.

La création d'une commande se trouve dans la section NEW-COMMANDE du programme (version 1).

Les paramètres en entrée pour la création d'une commande sont :

- ACCEPT-NCOM : N° de la commande
- ACCEPT-CLI : N° de client
- ACCEPT-DATE : Date de commande

```
CLIENT-EXIST.
 EXEC SQL
     SELECT NCLI
     into :NCLI-CLIENT
     FROM CLIENT
     WHERE NCLI =:ACCEPT-CLI
 END-EXEC
 .
```

Figure 6-4 Code de la vérification de l'existence du client.

```
INSERT-COMMANDE.
 EXEC SQL
      INSERT INTO COMMANDE(NCOM,CLI,DATECOM)
      VALUES (:ACCEPT-NCOM, :ACCEPT-CLI,:ACCEPT-DATE)
 END-EXEC
 .
```

Figure 6-5 Code d'insertion d'une commande.

```
* Check CLIENT DATA
       PERFORM CLIENT-EXIST
       PERFORM RESULT-READ

*INSERT DATA
       IF ROW-TROUVEE

          PERFORM  COMMANDE-EXIST

          IF NOT ROW-TROUVEE

           PERFORM INSERT-COMMANDE
           PERFORM RESULT-INSERT

           IF ROW-INSERT
            PERFORM DO-COMMIT
            DISPLAY "INSERT COMMANDE SUCCEED"
           ELSE
            DISPLAY "ERROR CREATING  COMMANDE"
           END-IF
          ELSE
              DISPLAY "ERROR INSERT COMMANDE : NCOM EXIST"
          END-IF
       ELSE
           DISPLAY "ERROR INSERT COMMANDE : NCLI NOT EXIST"
       END-IF
  END-IF
  .
```

Figure 6-6 Code de gestion des données pour une commande.

```
COMMANDE-EXIST.
 EXEC SQL
      SELECT NCOM
      into :NCOM-COM
      FROM COMMANDE
      WHERE NCOM =:ACCEPT-NCOM
 END-EXEC
 .
```

Figure 6-7 Code de la vérification de l'existence d'une commande

Dans la section NEW-COMMANDE (figure 6-6), on retrouve deux instances intéressantes de patterns.

La première se retrouve dans les figures 6-4, 6-5 et 6-6 et correspond à une instance du pattern numéro 10 de notre catalogue. En effet, la vérification de l'existence d'un numéro de client avant l'insertion d'un numéro de commande fait penser à l'existence d'une contrainte référentielle entre ses deux tables.
Ce pattern suggère donc l'hypothèse qu'il existe une clé étrangère de CLI de COMMANDE vers NCLI de CLIENT.
La seconde instance se retrouve dans les figures 6-5, 6-6 et 6-7. Elle fait penser à une instance du pattern n° 5 de notre catalogue qui suggère l'existence d'une clé primaire. En effet, il faut garantir l'unicité de l'entité commande avant son insertion.
Ce pattern introduit donc l'hypothèse que l'attribut NCOM est une clé primaire de COMMANDE. De plus, l'ordre de sélection de la figure 6-7 ne contient ni de DISTINCT, ni de boucle FETCH, ni de GROUP BY ou de fonction d'agrégat (patterns 1, 2, 3, 4 du catalogue) ce qui renforce un peu plus notre hypothèse.

Hypothèses émises :
- Clé étrangère de CLI de COMMANDE vers NCLI de CLIENT.
- NCOM clé primaire de COMMANDE.

b) Changer la valeur du n° d'un client.
Le changement d'un n° de client dans la table des clients se trouve dans la section UPDATE-NCLI-CLIENT du programme (version 1).

Les paramètres en entrée pour la mise à jour du n° de client sont :
- ACCEPT-NCLI-OLD : N° client à mettre à jour
- ACCEPT-NCLI-NEW : Nouveau n° de client

```
TEST-OLD-NCLI-EXIST.
    EXEC SQL
        SELECT NCLI
        into :NCLI-CLIENT
        FROM CLIENT
        WHERE NCLI =:ACCEPT-NCLI-OLD
    END-EXEC
    .

TEST-NEW-NCLI-EXIST.
    EXEC SQL
        SELECT NCLI
        into :NCLI-CLIENT
        FROM CLIENT
        WHERE NCLI =:ACCEPT-NCLI-NEW
    END-EXEC
    .
```

Figure 6-8 Code de vérification du n° de client.

```
UPDATE-NCLI-IN-CLIENT.
   EXEC SQL
       UPDATE CLIENT
       SET NCLI=:ACCEPT-NCLI-NEW
       WHERE NCLI =:ACCEPT-NCLI-OLD
   END-EXEC
   .
```

Figure 6-9 Code de mise à jour du n° de client dans la table CLIENT.

```
       PERFORM TEST-OLD-NCLI-EXIST
      PERFORM RESULT-READ

      IF ROW-TROUVEE
         PERFORM TEST-NEW-NCLI-EXIST
         PERFORM RESULT-READ

         IF NOT ROW-TROUVEE

*          UPDATE DATA IN TABLE CLIENT
           PERFORM UPDATE-NCLI-IN-CLIENT
           PERFORM RESULT-UPDATE

           IF ROW-UPDATE
*             UPDATE DATA IN TABLE COMMANDE
              PERFORM UPDATE-CLI-IN-COMMANDE
              PERFORM RESULT-UPDATE
              IF ROW-UPDATE
                PERFORM DO-COMMIT
                DISPLAY "UPDATE SUCCEED" ACCEPT-NCLI-OLD" TO"
                                         ACCEPT-NCLI-NEW
              ELSE
                DISPLAY "ERROR UPDATE NCLI IN COMMANDE"
              END-IF
           ELSE
             DISPLAY "ERROR UPDATE NCLI IN CLIENT"
           END-IF

         ELSE
           DISPLAY "ERROR UPDATE NCLI CLIENT, NB ALREADY EXIST"
         END-IF
      ELSE
          DISPLAY "ERROR UPDATE NCLI CLIENT, NB NOT EXIST"
      END-IF
      .
```

Figure 6-10 Code de gestion de la mise à jour d'un n° de client.

```
UPDATE-CLI-IN-COMMANDE.
   EXEC SQL
       UPDATE COMMANDE
       SET CLI=:ACCEPT-NCLI-NEW
       WHERE CLI =:ACCEPT-NCLI-OLD
   END-EXEC
```

Figure 6-11 Code de mise à jour du n° de client dans la table COMMANDE.

Dans la section UPDATE-NCLI-CLIENT (figure 6-10), on retrouve une instance intéressante du pattern numéro 11 de notre catalogue.

Ce pattern englobe le pattern n°5 que l'on retrouve dans les figures 6-8 et 6-9. Celui-ci vérifie l'existence de l'ancienne valeur du numéro de client et la non-existence du nouveau numéro de client et permet de poser l'hypothèse sur l'attribut NCLI supposant que celui-ci est une clé primaire de l'entité CLIENT. La figure 6-8 ne contient ni de DISTINCT, ni de boucle FETCH, ni de GROUP BY ou de fonction d'agrégat (patterns 1, 2, 3, 4 du catalogue) ce qui renforce un peu plus notre hypothèse.

La figure 6-11 vient compléter la fin du pattern numéro 11 du catalogue.

En effet, la mise à jour en cascade de la table commande suite à la mise à jour de la table CLIENT fait penser à l'existence d'une contrainte référentielle entre ses deux tables. Ce pattern introduit donc l'hypothèse qu'il existe une clé étrangère de CLI de COMMANDE vers NCLI de CLIENT.

Hypothèses émises :
- Clé étrangère de CLI de COMMANDE vers NCLI de CLIENT.
- NCLI clé primaire de CLIENT.

c) Affichage des commandes passées dans une localité donnée.

L'affichage des commandes passées dans une localité se trouve dans la section DISPLAY-COMMANDE-LOCALITE du programme (version1 et version2).

Le paramètre en input pour la consultation d'une commande est :
- ACCEPT-LOCA : localité du client

Dans la version 1 du programme (figure 6-12, 6-13 et 6-14) la lecture des tables CLIENT et COMMANDE se fait via la clause WHERE (figure 6-14).

```
*CURSOR
 EXEC SQL
          DECLARE C1-DISPLAY-LOCA CURSOR FOR
          SELECT B.NCOM, A.NCLI, A.NOM , A.LOCALITE
            FROM CLIENT A, COMMANDE B
           WHERE A.NCLI=B.CLI AND
                 A.LOCALITE =:ACCEPT-LOCA
 END-EXEC
```

Figure 6-12 Déclaration du curseur de sélection des commandes dans une localité donnée.

```
DISPLAY-DATA-LOCA.
    EXEC SQL
       FETCH C1-DISPLAY-LOCA INTO :NCOM-COM,
                                  :NCLI-CLIENT ,
                                  :NOM-CLIENT,
                                  :LOCA-CLIENT
    END-EXEC

    IF SQLCODE NOT = 100
       DISPLAY "N° COM : " NCOM-COM
               "NOM CLIENT : " NOM-CLIENT
               "LOCALITE   :" LOCA-CLIENT
    END-IF
    .
```

Figure 6-13 Code de gestion d'affichage des commandes dans une localité donnée.

```
*DISPLAY DATA

        EXEC SQL
           OPEN C1-DISPLAY-LOCA
        END-EXEC

        PERFORM DISPLAY-DATA-LOCA UNTIL SQLCODE = 100

        EXEC SQL
           CLOSE C1-DISPLAY-LOCA
        END-EXEC

    END-IF
    .
```

Figure 6-14 Affichage des données concernant les commandes

Dans la section DISPLAY-COMMANDE-LOCALITE (figure 6-13), on retrouve une instance intéressante de patterns.

Il s'agit d'une instance du pattern numéro 6 de notre catalogue. En effet, la présence de jointure via l'utilisation de clause WHERE (figure 6-14) fait penser qu'il existe une clé étrangère potentielle entre la table CLIENT et COMMANDE. Ce pattern introduit donc l'hypothèse qu'il existe une clé étrangère de CLI de COMMANDE vers NCLI de CLIENT. L'utilisation de cette jointure suppose également que NCLI est une clé primaire de CLIENT.

Hypothèses émises :
- Clé étrangère de CLI de COMMANDE vers NCLI de CLIENT.
- NCLI clé primaire de CLIENT.

Dans la version 2 du programme (figure 6-15, 6-16 et 6-17), la lecture des tables CLIENT et COMMANDE se fait via l'utilisation de deux boucles « fetch » imbriquées.

```
*CURSOR n°1
 EXEC SQL
          DECLARE C1-DISPLAY-LOCA CURSOR FOR
          SELECT A.NCLI, A.NOM , A.LOCALITE
            FROM CLIENT A
           WHERE A.LOCALITE =:ACCEPT-LOCA
 END-EXEC


*CURSOR n°2
 EXEC SQL
          DECLARE C2-DISPLAY-LOCA CURSOR FOR
          SELECT B.NCOM
            FROM COMMANDE B
           WHERE B.CLI =: NCLI-CLIENT
 END-EXEC
```

Figure 6-15 Déclaration des curseurs de sélection des commandes dans une localité donnée.

```
DISPLAY-DATA-LOCA-2.
    EXEC SQL
       FETCH C2-DISPLAY-LOCA INTO :NCOM-COM
    END-EXEC

    IF SQLCODE NOT = 100

        DISPLAY "N° COM : " NCOM-COM
                "NOM CLIENT : " NOM-CLIENT
                "LOCALITE   :" LOCA-CLIENT
    END-IF
    .
```

Figure 6-16 Affichage des données concernant les commandes.

```
DISPLAY-COMMANDE-LOCALITE.

         DISPLAY "DISPLAY DATA COMMANDE ? Y/N".
         ACCEPT  DISP-LOCA-FLAG.

         IF DISP-LOCA-FLAG = "Y"
           DISPLAY "Enter - LOCALITE :"
           ACCEPT  ACCEPT-LOCA

    *DISPLAY DATA
    *1ere boucle

             EXEC SQL
                OPEN C1-DISPLAY-LOCA
             END-EXEC

             PERFORM DISPLAY-DATA-LOCA-1 UNTIL SQLCODE = 100

             EXEC SQL
                CLOSE C1-DISPLAY-LOCA
             END-EXEC

         END-IF
         .


DISPLAY-DATA-LOCA-1.
    EXEC SQL
       FETCH C1-DISPLAY-LOCA INTO :NCLI-CLIENT ,
                                  :NOM-CLIENT,
                                  :LOCA-CLIENT
    END-EXEC

    IF SQLCODE NOT = 100

 *2eme boucle

        EXEC SQL
           OPEN C2-DISPLAY-LOCA
        END-EXEC

        PERFORM DISPLAY-DATA-LOCA-2 UNTIL SQLCODE = 100

        EXEC SQL
           CLOSE C2-DISPLAY-LOCA
        END-EXEC

    END-IF
    .
```

Figure 6-17 Code de gestion d'affichage des commandes dans une localité donnée.

Dans la section DISPLAY-COMMANDE-LOCALITE et DISPLAY-DATA-LOCA-1 (figure 6-17), on retrouve une instance intéressante du pattern numéro 9 de notre catalogue. En effet, la présence de jointure via l'utilisation de deux boucles « fetch » imbriquées indique qu'il existe une clé étrangère potentielle entre la table CLIENT et COMMANDE.
L'utilisation de la variable hôte NCLI-CLIENT en « output » (figure 6-17) et en « input » (figure 6-15) permet d'établir le lien entre les deux requêtes[55]. En effet, la variable hôte utilisée pour l'output de la première requête est la même que celle utilisée pour la seconde requête.

Ce pattern introduit donc l'hypothèse qu'il existe une clé étrangère de CLI de COMMANDE vers NCLI de CLIENT. L'utilisation de cette jointure peut également supposer que NCLI est une clé primaire de CLIENT.

Dans la figure 6-15, nous retrouvons deux instances du pattern numéro 2 de notre catalogue. La première instance peut indiquer que l'attribut LOCALITE n'est pas une clé primaire de CLIENT. La seconde peut indiquer que CLI n'est pas une clé primaire de COMMANDE.

Hypothèses émises :
- Clé étrangère de CLI de COMMANDE vers NCLI de CLIENT.
- NCLI clé primaire de CLIENT.
- CLI n'est pas une clé primaire de COMMANDE.
- LOCALITE n'est pas une clé primaire de CLIENT.

Cette analyse de pattern des différents fragments du code source permet de supposer l'existence des contraintes implicites suivante :
- NCLI clé primaire de CLIENT.
- NCOM clé primaire de COMMANDE.
- CLI de COMMANDE clé étrangère vers NCLI de CLIENT.

Le schéma enrichi suite à l'analyse des patterns est présenté dans la figure 6-18.

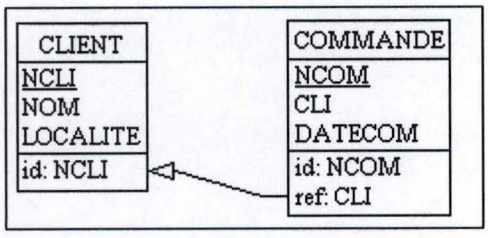


Figure 6-18 Schéma CLIENT-COMMANDE enrichi par l'analyse de patterns.

[55] output-intput dépendance [Cleve 2008].

6.6 Conclusion

Le but des patterns est de créer une sorte de corps de connaissances pour tenter d'aider l'ingénieur à identifier au mieux les contraintes implicites cachées dans les programmes applicatifs.
Le catalogue constitue une sorte de bibliothèque de connaissances dans lequel nous regroupons et communiquons notre savoir et notre expérience.

Bien que l'approche soit longue et complexe, les patterns permettent de clarifier certaines informations cachées dans le code source.

Certains articles montrent que cette voie d'exploration peut produire des résultats intéressants ([Petit 1994], [Petit 1995], [Signore 1994]).

Cependant, l'approche d'extraction révèle certains problèmes comme le risque d'analyser du code source mort, ce code n'étant donc plus à jour et pouvant conduire à des erreurs d'analyse ou à de mauvaises interprétations de résultats.

Néanmoins, cette phase de rétro-ingénierie reste nécessaire pour enrichir les informations extraites de la base de données.

Dans le chapitre relatant des perspectives futures, nous verrons comment l'analyse dynamique de requêtes peut pallier à ce problème. Nous expliquerons également comment une part significative de la charge de recherche peut être allégée en « traçant » l'activité du serveur et des programmes.

Chapitre 7

VALIDATION DES HYPOTHESES PAR CONFRONTATION

7.1 Introduction

Dans ce chapitre, nous illustrerons comment les hypothèses formulées sur base des analyses présentées dans les chapitres précédents peuvent être validées ou réfutées.

L'analyse de chacune des sources une à une s'avère insuffisante. Pour obtenir une vision globale du résultat final. Les différents éléments recueillis au cours des diverses phases d'analyse seront de ce fait comparés et confrontés pour former un tout cohérent (figure 7-1).

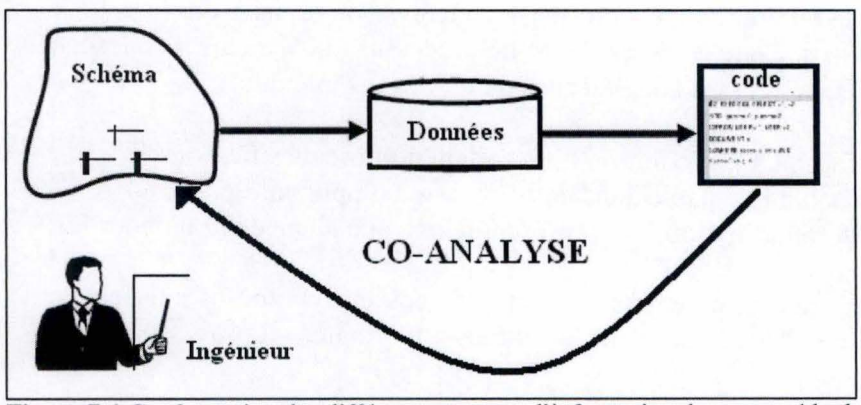


Figure 7-1 Confrontation des différentes sources d'information de notre méthodologie.

Après avoir collecté toute une série d'informations, l'ingénieur devra combiner les hypothèses et les indicateurs provenant des différentes sources d'information afin de les comparer.

En effet, toutes les méthodes d'élicitation seront utilisées pour répondre aux besoins spécifiques de la prise de décisions et valider ainsi les hypothèses de départ.

Divers indicateurs viendront apporter un poids aux différentes hypothèses afin d'en augmenter ou diminuer le pourcentage de cohérence.
En effet, l'ingénieur pourra associer un indice de confiance à chaque pièce de connaissance. Cette quantification des indicateurs permet de définir une mesure correspondant à un degré de validité pour chaque pièce de connaissance.
Par exemple nous pouvons imaginer que l'analyse des données ou du code source apporte un poids plus grand que l'analyse du schéma lors de la validation d'une hypothèse.

Cette confrontation est nécessaire car elle permet de pallier les incohérences dues à l'incomplétude ou à la non-exhaustivité des sources d'information disponibles et donc d'éviter de nombreux résultats incorrects.
En effet, plusieurs cas d'incohérences peuvent survenir, c'est pourquoi certaines phases du processus devront être ré-exécutées, ré-analysées ou simplement ignorées.

Par exemple, pour assurer l'hypothèse émise lors de l'analyse de schéma suggérant qu'il existe une clé étrangère entre le champ trouvé et un identifiant (contrainte référentielle), nous allons analyser les données et le(s) fragment(s) de code SQL en rapport avec l'instruction qui fait le lien entre les deux champs et si tous ces éléments (indices) concordent et en font un bon candidat pour être une clé étrangère, comme par exemple la présence d'une jointure entre ces deux attributs, nous confirmerons l'existence d'une contrainte référentielle potentielle entre ces deux champs.

Cette étape permettra également d'éliminer le bruit[56] et de réduire le silence[57].

Afin de lever les ambiguïtés résultant d'incohérences produites lors des différentes phases, l'ingénieur sera sollicité dans les étapes de décision concernant la validation ou la non-validation des contraintes identifiées. C'est lui qui validera et décidera s'il y a lieu de faire d'autres analyses, de réitérer certains processus d'analyse ou de faire des investigations plus poussées. Cette étape finale est un processus 100% manuel sollicitant l'expérience, les connaissances et les compétences du ou des ingénieurs.

L'ingénieur devra faire face et à la résolution d'un certain nombre de situations contradictoires, ce qui semble inévitable dans notre domaine d'application.
La confrontation des différentes hypothèses représente donc une tâche centrale pour lui.

Le but recherché est non seulement de réduire drastiquement les hypothèses « farfelues » mais également d'impliquer l'ingénieur dans les phases importantes du processus de validation.

Le processus final visera avant tout à enrichir et affiner le schéma de départ pour y incorporer les constructions implicites découvertes par les différentes analyses et confirmées grâce à la validation des différentes hypothèses.

[56] Le bruit est une contrainte détectée par une des phases d'analyse mais totalement inexistante.
[57] Le silence est une contrainte existante mais ignorée par une phase d'analyse.

7.2 Méthodologie

Au départ l'analyse de schéma émet une première hypothèse. Par après, une analyse plus large des autres sources d'information est établie.
L'analyse des données et des patterns peuvent révéler des indicateurs permettant de renforcer, de valider ou de réfuter cette hypothèse.

Pour chaque constructeur implicite, nous pouvons classifier les indices d'élicitation comme suit :

- Indices de formulation de l'hypothèse : Indice aidant à la formulation de l'hypothèse.
- Indices de validation de l'hypothèse : Indice augmentant l'assurance de l'hypothèse
- Indices de réfutation de l'hypothèse : Indice diminuant l'assurance de l'hypothèse.

Le « mot final » est à la charge de l'ingénieur; ce qui clôturera le processus d'analyse, c'est lui qui décidera de valider ou de réfuter l'hypothèse émise, et ce en fonction des divers indices collectés.

Les hypothèses validées seront alors converties en constructeur (code LDD, triggers, procédures, code applicatif) et le schéma sera enrichi.

Ce processus n'est pas sans failles, certaines contradictions ou inconsistances peuvent surgir, des erreurs peuvent toujours être présentes dans les programmes ou dans les données. En effet, durant la vie d'une application certaines contraintes peuvent être rajoutées ou enlevées et ces modifications peuvent ne pas avoir été répercutées sur l'ensemble des programmes ou des données. L'ingénieur peut également faire des erreurs d'interprétation au moment de valider ou de réfuter certaines hypothèses.

7.3 Illustration

Les différentes illustrations n'ont pas obligatoirement de cohérence entre elles, il s'agit simplement d'exemples significatifs servant à montrer le besoin des différentes sources d'information et pourquoi la confrontation est un processus primordial dans nos étapes d'élicitation.

Cette section illustre notre méthodologie au travers d'exemples montrant l'élicitation d'une clé étrangère.

Nous allons rechercher les liens entres attributs à l'aide de plusieurs techniques coordonnées.

7.3.1 Première illustration

Pour la phase de découverte des hypothèses, nous nous baserons sur le schéma de la figure 7-2 dans lequel deux tables semblent être liées par une contrainte référentielle. A noter qu'au départ nous partons de l'hypothèse que les identifiants de ces deux tables sont connus. Nous nous baserons également sur un fragment du code source (figure 7-3) du programme de gestion des clients et des commandes (illustration du chapitre 6) dans le but d'identifier une instance de pattern issu de notre catalogue.

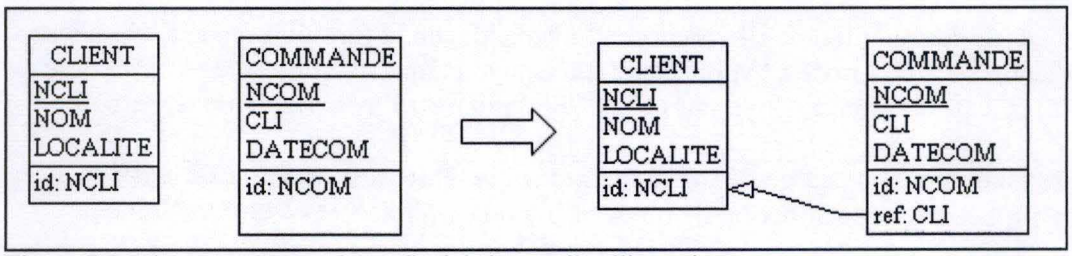


Figure 7-2 Schéma source et schéma final de la première illustration.

```
*CURSOR
 EXEC SQL
          DECLARE C1-DISPLAY-LOCA CURSOR FOR
          SELECT B.NCOM, A.NCLI, A.NOM , A.LOCALITE
            FROM CLIENT A, COMMANDE B
           WHERE A.NCLI=B.CLI AND
                 A.LOCALITE =:ACCEPT-LOCA
 END-EXEC
```

Figure 7-3 Fragment de code source utilisé pour la première illustration.

```
SELECT count(*) FROM COMMANDE where CLI not in (select NCLI from CLIENT);
```

Figure 7-4 Requête générée dans le but de vérifier l'existence potentielle d'une contrainte d'intégrité référentielle pour la première illustration.

Pour la phase de validation, nous analyserons les données. Le contenu de la base sera vérifié par une ou plusieurs requêtes SQL générées automatiquement via nos plugins.

La figure 7-4 montre la requête qui a été générée suite à cette analyse. Celle-ci compte le nombre d'enregistrements violant la contrainte référentielle. Si le nombre d'enregistrements est 0 la contrainte référentielle peut être validée. Dans le cas contraire, elle est réfutée. Bien sûr cette situation paraît un peu idéaliste car dans les bases de données réelles, il n'est pas rare de trouver des erreurs dans les données, mais si ce pourcentage d'erreurs est faible par rapport au volume des données, l'ingénieur pourra valider la contrainte référentielle.

Phase	Heuristiques	Description
Découverte hypothèse(s)	Analyse du schéma : Même type, même taille, nom identique à 80%. Analyse des patterns : Instance du pattern n°6 identifiée.	L'examen du nom, de la taille et du type nous donne une première indication sur l'existence d'une clé étrangère potentielle entre COMMANDE.CLI et CLIENT.CLI. L'examen du code source (figure 7-3) montre l'utilisation d'une jointure entre COMMANDE.CLI et CLIENT.CLI. Ce qui donne une deuxième indication sur l'existence d'une clé étrangère potentielle entre ses deux attributs.
Hypothèse(s)	COMMANDE.CLI >> CLIENT.CLI	CLI de COMMANDE est une clé étrangère vers NCLI de CLIENT.
Approbation hypothèse(s)	Analyse du schéma : 80% de similarité entre les noms, même taille, même type. Usage de patterns : Analyse des données :	CLI de COMMANDE et NCLI de CLIENT sont du même type, ont la même taille et leurs noms sont identiques à 80%. Instance du pattern n°6 utilisée. La requête validant l'hypothèse est illustrée dans la figure 7-4. Celle-ci a retourné 0 enregistrement et confirme la présence potentielle d'une clé étrangère entre COMMANDE.CLI et CLIENT.CLI.
Réfutation hypothèse(s)	/	Aucun indice n'est venu contredire l'hypothèse émise.

Sur base de ces 3 indices, nous pouvons en confiance conclure que CLI de COMMANDE peut être une clé étrangère vers NCLI de CLIENT, ce qui conduit au schéma augmenté de la figure 7-2.

7.3.2 Seconde illustration

Dans la seconde illustration nous sommes parti d'un exemple imaginé par [Barbar 2002]. Pour réaliser une petite étude concrète sur celui-ci nous avons créé et alimenté les tables correspondantes à cet exemple (figure 7-5).

Pour la phase de découverte des hypothèses, nous nous baserons sur le schéma de la figure 7-5 et sur un fragment du code source (figure 7-6). Dans ce fragment de code le programmeur a voulu identifier les voitures ayant un nom de fleur. Pour ce faire, il a utilisé une jointure entre les deux tables, ce qui va conduire l'analyse des patterns et l'analyse du schéma vers une piste erronée.

Pour la phase validation nous analyserons les données. Le contenu de la base sera vérifié par les requêtes SQL générées automatiquement via nos plugins. La figure 7-7, montre les requêtes générées suite à cette analyse.

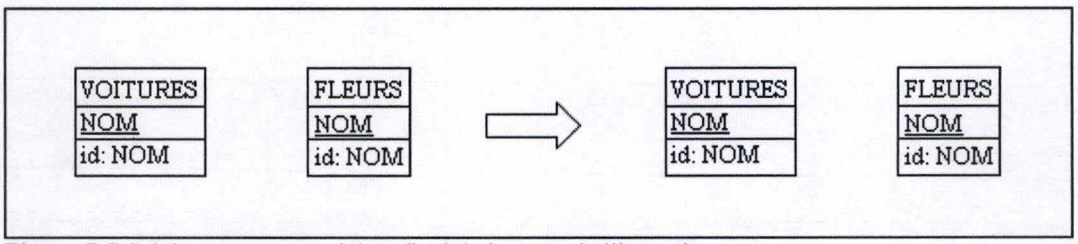


Figure 7-5 Schéma source et schéma final de la seconde illustration.

```
SELECT V.nom
FROM FLEURS F, VOITURES V
WHERE F.nom = V.nom;
```

Figure 7-6 Fragment de code source utilisé pour la seconde illustration.

```
-- SQL order to check Foreign Key : NOM of FLEURS to NOM of VOITURES
-- if result = 0, maybe potential FK (ref)
SELECT count(*) FROM FLEURS where NOM not in (select NOM from VOITURES);

-- SQL order to check Foreign Key : NOM of VOITURES to NOM of FLEURS
-- if result = 0, maybe potential FK (ref)
SELECT count(*) FROM VOITURES where NOM not in (select NOM from FLEURS);
```

Figure 7-7 Requête générée dans le but de vérifier l'existence potentielle d'une contrainte d'intégrité référentielle pour la seconde illustration.

Phase	Heuristiques	Description
Découverte hypothèse(s)	Analyse du schéma : Même type, même taille, nom identique. Analyse des patterns : Instance du pattern n°6 identifiée.	L'examen du nom, de la taille et du type nous donne une première indication sur l'existence de deux clés étrangères potentielles entre les tables FLEURS et VOITURES. L'examen du code source (figure 7-6) montre l'utilisation d'une jointure entre les tables VOITURES et FLEURS, ce qui donne une deuxième indication sur l'existence d'une clé étrangère potentielle entre ses deux attributs.
Hypothèse(s)	VOITURES.NOM >> FLEURS.NOM FLEURS.NOM >> VOITURES.NOM	NOM de VOITURES est une clé étrangère vers NOM de FLEURS. NOM de FLEURS est une clé étrangère vers NOM de VOITURES.
Approbation hypothèse(s)	Analyse du schéma : Même nom, même taille, même type. Usage de patterns :	NOM de VOITURES et NOM de FLEURS sont du même type, ont la même taille et leurs noms sont identiques. Instance du pattern n°6 utilisé.
Réfutation hypothèse(s)	Analyse des données :	Les requêtes ayant permis de réfuter les deux hypothèses sont illustrées dans la figure 7-7. Le résultat de ces requêtes a retourné un nombre différent de 0, ce qui peut fortement confirmer la non présence d'une clé étrangère entre les tables FLEURS et VOITURES.

Sur base du résultat de l'analyse des données, l'ingénieur à décidé de conclure qu'il n'y avait pas de clés étrangères entre les tables FLEURS et VOITURES. En effet, celui-ci a accordé un indice de confiance dans l'analyse des données supérieur à celui de l'analyse du schéma et des patterns.

Des résultats concluants ne peuvent être obtenus seulement en analysant l'information du schéma et des patterns. Le scannage des données (chapitre 5) est un processus plus que nécessaire.

7.4 *Conclusion*

Dans ce processus, l'ingénieur confirme ou réfute les hypothèses incertaines et résout les contradictions pour obtenir des résultats cohérents.

Cependant, cette phase d'approbation ou de réfutation des hypothèses est en partie manuelle dans notre approche et utiliser ce mécanisme dans des applications plus larges peut s'avérer être un travail fastidieux.

Mais ce processus de validation reste avant tout un processus de décision. Il ne serait être formel ou déterministe; l'implication de l'ingénieur, son expérience et ses connaissances du domaine constituent un atout capital pour assurer la réussite et la cohérence de cette phase.

Le but recherché ici était avant tout de montrer que l'imbrication des différents processus constitue une tâche fondamentale dans le processus de rétro-ingénierie.

Chapitre 8

AMELIORATION DE LA QUALITE

8.1 Introduction

La rétro-ingénierie des bases de données, en reconstituant une partie ou l'ensemble des règles de gestion du système d'information, permet d'obtenir des modèles de base de données plus robustes; elle va renforcer ou (re)construire les « murs ».

En effet, une contrainte implicite non vérifiée laisse la base de données vulnérable face à la corruption de données.

Les progrès dans les langages de programmation, dans la technologie des bases de données et dans les connaissances montrent que la vérification des contraintes peut être renforcée.

La (re)constitution des clés, des contraintes d'intégrité, des déclencheurs (ou procédures) et la correction du code améliore de manière significative la qualité du système face aux risques de violation d'une ou des contrainte(s).

De la même façon, la gestion et les tâches d'administration de la base de données peuvent en être simplifiées et réduites.

Dans ce chapitre nous expliquerons avant tout ce que nous entendons par le mot « qualité » ou plutôt « qualité de la base de données ».

Le mot « qualité » couplé au mot « donnée » tel que nous l'utiliserons inclut les mots justesse, complétude, consistance et crédibilité ([Milano 2005], [Cleve 2008a]).

Notre étude mettra en avant la qualité des données et sera focalisée tout d'abord sur la détection des incohérences, pour ensuite se pencher sur le contrôle de la propagation des erreurs et sur le renforcement de la validation des contraintes, tant du côté de l'application que du côté de la base de données.

Un des buts de nos recherches est de faire évoluer le système vers un niveau de qualité supérieur en y apportant diverses solutions dont :

- Un analyseur de données permettant de détecter les éventuelles données invalides par rapport à une contrainte.

- Un générateur de contrôles (triggers) garantissant le respect de la contrainte lors des opérations de mise à jour (insert, delete, update).

- Un catalogue de patterns dont la consultation pourra guider l'ingénieur vers une correction éventuelle des requêtes contenues dans les modules sources. En effet, celui-ci apporte une couche supplémentaire de vérification entre le programme et la base de données.

La mise en place de ces mécanismes de contrôle aura pour but de « stopper l'hémorragie ».

Ce renforcement, tant du côté de la base de données que du côté programme, réduira considérablement le risque d'introduction de données incohérentes dans la base de données et pourra garantir une stabilité plus grande de l'application.

L'expérience montre bien souvent que les « abends[58] » dans les programmes applicatifs surviennent la plupart du temps lorsque l'application comporte une ou plusieurs incohérences dans les données.

Le but principal de ce chapitre est de se consacrer à la détection des données « erronées » et à la mise en place de « barrages » visant à stopper la propagation de ces erreurs en utilisant comme point de départ les contraintes identifiées à partir des techniques d'analyses définies dans les chapitres précédents.

Cet enjeu est double : premièrement, renforcer la consistance du système en s'assurant du respect des contraintes implicites ayant été découvertes par nos analyses. Secondement, garantir que chaque contrainte est bien gérée tant au niveau de la base de données qu'au niveau de l'application.

A noter que le « nettoyage » des erreurs subsistantes dans la base de données et la modification du code ne fait pas partie de l'objectif de ce travail. Celles-ci sont sous la responsabilité de l'ingénieur.

8.2 La qualité

La qualité d'une base de données découle bien souvent de la qualité de son système d'information. Sans informations fiables, pertinentes, intègres et cohérentes, il est difficile d'agir au quotidien et le système d'information peut très vite devenir obsolète et instable.

La qualité des données et des bases de données est gérée par les programmes et par le système de gestion de bases de données, elle relève donc des techniques informatiques.

[58] Abnormal End, soit « arrêt anormal ».

Il n'est pas rare d'être confronté à des données erronées ou contradictoires.
L'exécution d'un programme peut également engendrer un « abend » suite à des données incohérentes.

Mon expérience professionnelle m'a souvent montré que ce genre de situations étaient nuisibles à l'image interne (insatisfaction de la direction suite aux dépassements de budgets et au retard pris par le projet, etc.) et externe (clients mécontents, etc.) de l'entreprise.

La qualité des informations a donc un impact direct sur les coûts de l'entreprise et sur la satisfaction des clients. Celle-ci est devenue au fil des années un enjeu majeur pour le business de l'entreprise.

C'est dans cet état d'esprit que nous avons décidé de nous pencher un peu plus sur le renforcement de la qualité.

8.3 Renforcement de la qualité

Dans cette section nous avons envisagé de renforcer la qualité des données tant du côté de la base de données que du côté des programmes.

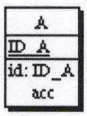
8.3.1 Renforcement côté base de données

Dans cette approche nous décrirons deux mécanismes permettant de renforcer la qualité de la base de données.

Le premier mécanisme tombe dans la catégorie de validation des contraintes.
Celui-ci tentera de renforcer la base de données via des mécanismes de contrôle et d'alerte appelés « triggers ». Ceux-ci permettront de vérifier que les données respectent les règles du modèle et limiteront le risque d'introduction de données inconsistantes dans la base.

Pour ce faire notre générateur créera un trigger pour chaque contrainte potentielle identifiée dans l'analyse du schéma. Celui-ci pourra alors être intégré dans le SGDB afin de garantir le respect de la contrainte lors de l'exécution d'opérations critiques (insert, delete, update) et sera déclenché dès que le SGBD tentera toutes modifications risquant de violer cette contrainte.

La figure 8-1 montre les triggers qui ont été générés afin d'assurer le respect de la contrainte d'unicité de la clé primaire de la table A.




```
-- creation trigger on INSERT for Primary Key on A
create trigger CHK_PK_IN_A
BEFORE INSERT ON A
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM A WHERE ID_A=new.ID_A) THEN
     call my_error_pk_violation();
END IF;
END

-- creation trigger on UPDATE for Primary Key on A
create trigger CHK_PK_UP_A
BEFORE UPDATE ON A
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM A WHERE ID_A=new.ID_A) THEN
  IF ((old.ID_A <> new.ID_A)) THEN
     call my_error_pk_violation();
  END IF;
END IF;
END
```

Figure 8-1 Triggers garantissant la contrainte d'unicité de la clé primaire.

Les figures 8-2 et 8-3 montrent les triggers qui ont été générés afin d'assurer le respect de la contrainte d'intégrité référentielle entre la table A et la table B.

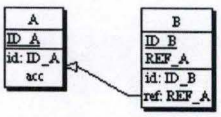
La gestion de la contrainte d'intégrité référentielle est complexe. Elle doit maintenir les liens entre les tables quelles que soient les modifications engendrées sur les données de la table parent ou de la table enfant.

Elle doit assurer lors de la mise à jour (UPDATE) ou lors de la suppression (DELETE) d'une ligne de la table parent que :

- La référence de la ligne parent, si elle est modifiée, soit répercutée dans toutes les lignes des tables enfants qui la référencent ou alors que toute modification de cette référence soit interdite si des lignes de tables enfants l'utilise.
  Les lignes enfants qui référencent la ligne mise à jour de la table parent peuvent également prendre une valeur par défaut ou une valeur nulle.

- Toute ligne référencée par une autre table ne soit pas supprimée, ou alors que l'on supprime aussi toutes les lignes enfants des tables qui référencent la ligne supprimée de la table parent.
  Les lignes enfants qui référencent la ligne supprimée de la table parent peuvent également prendre une valeur par défaut ou une valeur nulle.

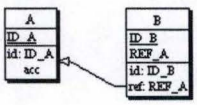
Pour la génération des triggers au niveau de la table parent nous avons pris l'option la plus saine qui est de générer un blocage en cas de mise à jour ou de suppression de la référence d'une ligne parent si celle-ci est référencée dans une ligne enfant (figure 8-3).




```
-- creation trigger on INSERT for Foreign Key (child): REF_A of B
create trigger CHK_FK_CHILD_IN_REF_A_B
BEFORE INSERT ON B
FOR EACH ROW
BEGIN
IF NOT (new.REF_A is null) THEN
 IF NOT EXISTS (SELECT ID_A FROM A WHERE ID_A= new.REF_A) THEN
     call my_error_fk_violation();
 END IF;
END IF;
END

-- creation trigger on UPDATE for Foreign Key (child): REF_A of B
create trigger CHK_FK_CHILD_UP_REF_A_B
BEFORE UPDATE ON B
FOR EACH ROW
BEGIN
IF NOT (new.REF_A is null) THEN
 IF (old.REF_A <> new.REF_A) THEN
  IF NOT EXISTS (SELECT ID_A FROM A WHERE ID_A= new.REF_A) THEN
     call my_error_fk_violation();
  END IF;
 END IF;
END IF;
END
```

Figure 8-2 Triggers assurant la gestion de l'intégrité référentielle au niveau de la table enfant.




```
-- creation trigger on DELETE  in parent table for Foreign Key : REF_A of B
create trigger CHK_FK_PARENT_DEL_ID_A_A
BEFORE DELETE ON A
FOR EACH ROW
BEGIN
 IF EXISTS (SELECT REF_A FROM B WHERE REF_A= old.ID_A) THEN
     call my_error_fk_violation();
 END IF;
END


-- creation trigger on UPDATE  in parent table for Foreign Key : REF_A of B
create trigger CHK_FK_PARENT_UPD_ID_A_A
BEFORE UPDATE ON A
FOR EACH ROW
BEGIN
IF (old.ID_A <> new.ID_A) THEN
  IF EXISTS (SELECT REF_A FROM B WHERE REF_A= old.ID_A) THEN
     call my_error_fk_violation();
  END IF;
END IF;
```

Figure 8-3 Triggers assurant la gestion de l'intégrité référentielle au niveau de la table parent.

Les triggers permettent de renforcer les contraintes dans le but d'assurer l'intégrité et la cohérence des données. Sans ces contraintes, une base de données est rapidement incohérente et faiblement intègre, la redondance peut y être commune.
Ces contraintes permettent également de rendre plus aisé, rapide et efficace le traitement des données.

De plus amples informations sur les triggers peuvent être trouvées dans [Hainaut 2006].

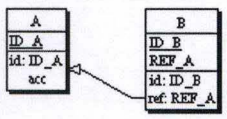
Le second mécanisme tombe dans la catégorie de détection des données « erronées ».
Pour tenter de détecter les données invalides par rapport à une contrainte, notre analyseur de données générera automatiquement des requêtes de détection. Son objectif est donc d'identifier les données violant une contrainte potentielle.

La figure 8-4 montre un exemple de requêtes générées pour détecter les enregistrements violant une contrainte d'unicité et d'existence. La première requête recherche toutes les instances de la table présentant des doublons et la seconde recherche les instances ayant une valeur à nulle.


```
-- check rows with no uniqueness
SELECT * FROM A where ID_A IN (select ID_A from A group by ID_A having count(ID_A) > 1 );
-- check rows with null
SELECT * FROM A where ID_A is null;
```

Figure 8-4 Exemple de requêtes générées pour détecter les enregistrements violant une contrainte d'unicité et d'existence.

La figure 8-5 montre un exemple de requête générée pour détecter les contraintes référentielles erronées. Celle-ci recherche toutes les instances de la table enfant n'étant pas référencées dans la table parent.




```
-- SQL order to check error on Foreign Key : REF_A of B to ID_A of A
SELECT * FROM B where REF_A not in (select ID_A from A);
```

Figure 8-5 Exemple d'une requête générée pour détecter les contraintes référentielles erronées.

Si l'une des requêtes générée pour détecter les données inconsistantes donne un résultat, les contraintes sont violées. Dans ce cas, deux solutions sont envisageables :

- La modification des données est ignorée car non pertinente.
- La modification des données est réalisée.

Dans ce travail, la correction des données erronées est à la charge de l'ingénieur.
En effet, malgré que la recherche des données invalides soit un processus automatique, leur résolution reste du domaine de l'ingénieur. Bien souvent, c'est au niveau de la logique business que se décidera la solution finale à apporter à ces données (correction, suppression, ne rien faire, etc.). Cette solution manuelle de correction est de loin la plus précise et la plus fiable.

A noter que les instances des données erronées peuvent également être transférées dans une table temporaire. Cette situation a été analysée dans [Hick 2001].

8.3.2 Renforcement côté programme

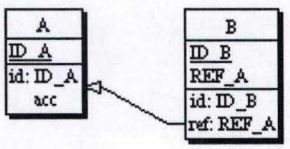
Cette stratégie se focalise sur la gestion et la validation des contraintes au niveau des données dans les programmes applicatifs. En théorie, les programmes contiennent les contraintes n'ayant pas été gérées via le système de gestion de base de données.

En effet, il y a de nombreux cas où les contraintes se retrouvent gérées dans le code applicatif. Les anciens systèmes de gestion de base de données étaient peu expressifs,

n'offrant pas la possibilité de « trigger » ou de définir certains types de contraintes comme les clés primaires ou les contraintes d'intégrité référentielle. Celles-ci se retrouvent donc gérées dans les programmes de différentes façons (gestion centralisée par un module de contrôle contrôlant toute mise à jour des tables; gestion distribuée où chaque module est responsable de la gestion des contraintes que son comportement pourrait violer; interfaces utilisateur garantissant le respect des contraintes, etc.).

Le but de cette section est d'identifier à l'aide de notre catalogue de patterns les modules sources contenant des opérations dites « critiques », c'est à dire les requêtes pouvant mettre potentiellement en danger l'intégrité et la cohérence des données et du système.

La figure 8-6 montre un de ces patterns assurant l'intégrité référentielle lors de la mise à jour de la table enfant. Dans les programmes, chaque instruction mettant à jour une table enfant doit s'assurer de la gestion de sa contrainte référentielle. En effet, chaque fois qu'un enregistrement est insérer ou updater dans la table enfant, une vérification doit être opérée pour vérifier l'existence des champs référencés dans la table parent. Il en est de même lors de la modification d'une table parent.
Il s'agit donc d'une validation réactive assurant que la contrainte est satisfaite avant l'exécution de la requête de mise à jour.




```
EXEC SQL BEGIN TRANSACTION;

EXEC SQL
SELECT *
FROM A
WHERE ID_A = :zone;

if (SQLCODE == 0)
{
  EXEC SQL
  INSERT INTO B (ID_B, REF_A)
  VALUES (:b, :zone);
  EXEC SQL COMMIT WORK;

}
else <call of the error-handling routine>
```

Figure 8-6 Base de données abstraite et son pattern vérifiant la contrainte d'intégrité référentielle.

L'objectif ici est d'identifier les fragments de code pour lesquels une contrainte aurait été mal gérée. Cette approche manuelle est une tâche longue et complexe. Elle est avant tout basée sur l'interprétation et le feeling des ingénieurs.
Le rôle de notre catalogue permet de les guider dans la recherche des éventuelles requêtes ou fragments de code dit « suspects ».

En ce qui concerne la modification de programme, le principe adopté ici consiste essentiellement à localiser et à donner une indication sur les parties de programmes où des modifications pourraient être effectuées afin de renforcer la qualité.

La modification de code a toujours été loin d'être triviale. C'est une tâche complexe et a priori non automatisable, sauf dans des situations simples où les modifications sont

mineures. En général, la modification des programmes est laissée à la responsabilité des programmeurs. Ce sont eux qui réaliseront les changements nécessaires là où les requêtes « non saines » ont été localisées.
Le but recherché ici étant avant tout de signaler la présence d'un danger potentiel dans le code source lors de la gestion d'une contrainte.

8.4 Illustration

Par cette illustration, nous avons voulu montrer à travers un exemple intuitif, l'importance et l'utilité de chacun de ces mécanismes de protection.

Pour ce faire nous sommes reparti de l'exemple de la gestion des clients et des commandes présenté dans le chapitre 6 et dont le code source se trouve en annexe.

Le schéma de départ est présenté dans la figure 8-7. L'analyse du schéma a permis de mettre à jour certaines contraintes implicites. Deux clés primaires et une contrainte d'intégrité référentielle ont été élicitées.

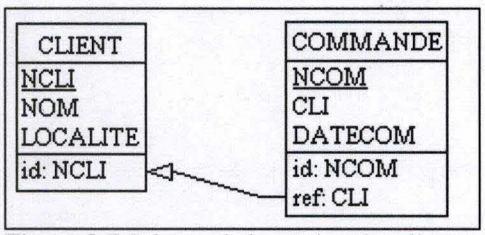


Figure 8-7 Schéma de la gestion des clients et des commandes.

Les figures 8-8 et 8-9 donnent un aperçu des triggers générés à partir de ces contraintes. Ceux-ci permettent de garantir, d'une part l'unicité de la clé primaire de la table CLIENT et COMMANDE et d'autre part l'intégrité référentielle de CLI de COMMANDE vers NCLI de CLIENT.

```
-- creation trigger on INSERT for Primary Key on CLIENT
create trigger CHK_PK_IN_CLIENT
BEFORE INSERT ON CLIENT
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM CLIENT WHERE NCLI=new.NCLI) THEN
     call my_error_pk_violation();
END IF;
END


-- creation trigger on UPDATE for Primary Key on CLIENT
create trigger CHK_PK_UP_CLIENT
BEFORE UPDATE ON CLIENT
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM CLIENT WHERE NCLI=new.NCLI) THEN
  IF ((old.NCLI <> new.NCLI)) THEN
     call my_error_pk_violation();
  END IF;
END IF;
END
```

```
-- creation trigger on INSERT for Primary Key on COMMANDE
create trigger CHK_PK_IN_COMMANDE
BEFORE INSERT ON COMMANDE
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM COMMANDE WHERE NCOM=new.NCOM) THEN
     call my_error_pk_violation();
END IF;
END


-- creation trigger on UPDATE for Primary Key on COMMANDE
create trigger CHK_PK_UP_COMMANDE
BEFORE UPDATE ON COMMANDE
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM COMMANDE WHERE NCOM=new.NCOM) THEN
  IF ((old.NCOM <> new.NCOM)) THEN
     call my_error_pk_violation();
  END IF;
END IF;
END
```

Figure 8-8 Triggers garantissant la contrainte d'unicité des clés primaires des tables CLIENT et COMMANDE.

```
-- creation trigger on INSERT for Foreign Key (child): CLI of COMMANDE
create trigger CHK_FK_CHILD_IN_CLI_COMMANDE
BEFORE INSERT ON COMMANDE
FOR EACH ROW
BEGIN
IF NOT (new.CLI is null) THEN
 IF NOT EXISTS (SELECT NCLI FROM CLIENT WHERE NCLI= new.CLI) THEN
     call my_error_fk_violation();
 END IF;
END IF;
END

-- creation trigger on UPDATE for Foreign Key (child): CLI of COMMANDE
create trigger CHK_FK_CHILD_UP_CLI_COMMANDE
BEFORE UPDATE ON COMMANDE
FOR EACH ROW
BEGIN
IF NOT (new.CLI is null) THEN
 IF (old.CLI <> new.CLI) THEN
  IF NOT EXISTS (SELECT NCLI FROM CLIENT WHERE NCLI= new.CLI) THEN
     call my_error_fk_violation();
  END IF;
 END IF;
END IF;
END

-- creation trigger on DELETE  in parent table for Foreign Key : CLI of COMMANDE
create trigger CHK_FK_PARENT_DEL_NCLI_CLIENT
BEFORE DELETE ON CLIENT
FOR EACH ROW
BEGIN
 IF EXISTS (SELECT CLI FROM COMMANDE WHERE CLI= old.NCLI) THEN
     call my_error_fk_violation();
 END IF;
END

-- creation trigger on UPDATE  in parent table for Foreign Key : CLI of COMMANDE
create trigger CHK_FK_PARENT_UPD_NCLI_CLIENT
BEFORE UPDATE ON CLIENT
FOR EACH ROW
BEGIN
IF (old.NCLI <> new.NCLI) THEN
  IF EXISTS (SELECT CLI FROM COMMANDE WHERE CLI= old.NCLI) THEN
     call my_error_fk_violation();
  END IF;
END IF;
END
```

Figure 8-9 Triggers assurant la gestion de l'intégrité référentielle au niveau de la table COMMANDE et CLIENT.

Les requêtes SQL illustrées dans les figures 8-10 et 8-11 permettent de vérifier que la base de données ne contient pas des données erronées par rapport aux contraintes identifiées.

```
-- SQL order to check data error on Primary Key NCLI on CLIENT
-- check rows with no uniqueness
SELECT * FROM CLIENT where NCLI IN (select NCLI from CLIENT group by NCLI having count(NCLI) > 1 );
-- check rows with null
SELECT * FROM CLIENT where NCLI is null;

-- SQL order to check data error on Primary Key NCOM on COMMANDE
-- check rows with no uniqueness
SELECT * FROM COMMANDE where NCOM IN (select NCOM from COMMANDE group by NCOM having count(NCOM) > 1 );
-- check rows with null
SELECT * FROM COMMANDE where NCOM is null;
```

Figure 8-10 Requêtes générées pour détecter les enregistrements violant la contrainte d'unicité et d'existence des clés primaires des tables CLIENT et COMMANDE.

```
-- SQL order to check error on Foreign Key : CLI of COMMANDE to NCLI of CLIENT
SELECT * FROM COMMANDE where CLI not in (select NCLI from CLIENT);
```

Figure 8-11 Requête générée pour détecter les enregistrements violant la contrainte référentielle entre la table CLIENT et COMMANDE.

La figure 8-12 donne un exemple de fragment de code montrant la bonne gestion de la contrainte d'intégrité référentielle. Avant d'insérer une ligne dans la table des COMMANDE, le programme vérifie que la valeur de la variable ACCEPT-CLI existe bien dans la table de référence des CLIENT.

```
* Check CLIENT DATA
       PERFORM CLIENT-EXIST
       PERFORM RESULT-READ

*INSERT DATA
       IF ROW-TROUVEE

          PERFORM  COMMANDE-EXIST

          IF NOT ROW-TROUVEE

           PERFORM INSERT-COMMANDE
           PERFORM RESULT-INSERT

           IF ROW-INSERT
            PERFORM DO-COMMIT
            DISPLAY "INSERT COMMANDE SUCCEED"
           ELSE
            DISPLAY "ERROR CREATING  COMMANDE"
           END-IF
          ELSE
              DISPLAY "ERROR INSERT COMMANDE : NCOM EXIST"
          END-IF
       ELSE
          DISPLAY "ERROR INSERT COMMANDE : NCLI NOT EXIST"
       END-IF
  END-IF
  .
```

```
CLIENT-EXIST.
 EXEC SQL
     SELECT NCLI
     into :NCLI-CLIENT
     FROM CLIENT
     WHERE NCLI =:ACCEPT-CLI
 END-EXEC
 .
```

```
INSERT-COMMANDE.
 EXEC SQL
      INSERT INTO COMMANDE(NCOM,CLI,DATECOM)
      VALUES (:ACCEPT-NCOM, :ACCEPT-CLI,:ACCEPT-DATE)
 END-EXEC
 .
```

Figure 8-12 Pattern illustrant une validation réactive assurant que la contrainte d'intégrité référentielle est satisfaite avant l'exécution de la requête de mise à jour.

8.5 *Conclusion*

Le but de ce chapitre était de montrer comment on pouvait apporter des solutions concrètes pour améliorer et renforcer la qualité des données à partir des contraintes implicites identifiées dans nos analyses précédentes.

Les mécanismes de renforcement ont été mis en place tant du côté de la base de données que du côté du code source.
Ceux-ci étaient principalement basés sur le traitement ou la mise à jour par l'ingénieur des données inconsistantes détectées grâce aux requêtes générées par notre analyseur de données, sur la mise en place de triggers générés automatiquement par notre générateur de requêtes de contrôle et sur la correction manuelle des requêtes dites « suspectes » suite à l'inspection visuelle des modules sources à l'aide de notre catalogue de patterns.

L'avantage de notre approche est double. Dans un premier temps, elle instaure une double couche de protection pour réduire l'introduction de données erronées dans le système d'information afin de s'assurer qu'en aucun cas une contrainte ne peut être violée. Dans un second temps, elle identifie les données inconsistantes de la base de données.

Bien entendu, il existe d'autres solutions pouvant venir enrichir la qualité du système comme les procédures stockées, l'utilisation du mot clé CHECK dans le code LDD, les programmes et interfaces de contrôle, les vues, etc.

L'objectif recherché était avant tout de montrer que nos mécanismes de contrôle pouvaient contribuer à l'amélioration de la qualité et constituer un pas important et utile dans l'amélioration du système.

Chapitre 9

CONCLUSION ET PERSPECTIVES FUTURES

9.1 Conclusion

Ce chapitre résume la contribution du mémoire et ses limites sur l'élicitation des contraintes implicites dans le domaine des bases de données relationnelles.

L'approche proposée dans notre travail était basée sur les problèmes de rétro-ingénierie de bases de données relationnelles englobant le schéma, les données et les programmes. Notre objectif était de considérer l'ensemble de ces problèmes de façon globale, puis de proposer des solutions efficaces pour leur résolution.

Le but recherché était donc de démontrer que ce mémoire avait du sens et que nos recherches et outils étaient viables.

Un processus de rétro-ingénierie a donc été présenté et illustré, la récupération d'une partie de la sémantique a été réalisée en partant de l'analyse du schéma et en y imbriquant par la suite le résultat de l'analyse et de l'étude d'autres sources d'information.
La caractéristique principale de cette approche était d'utiliser plusieurs sources d'information combinées.

Au niveau de l'analyse du schéma, les attributs par leurs caractéristiques ont été mis en corrélation. Dans l'analyse des données ces corrélations ont été vérifiées.
Les patterns ont joué un rôle à la fois d'examinateur et de vérificateur.

Le raffinement du schéma est parti d'un processus itératif dans lequel le schéma a servi de point d'entrée pour l'émission d'hypothèses sur les contraintes potentielles, lesquelles ont été enrichies, validées ou réfutées par les données et les patterns.

La conception de plugins illustrant nos différentes approches a constitué un challenge important pour illustrer et donner du sens à nos recherches.

Le mémoire a contribué à montrer que le succès d'un projet de rétro-ingénierie est avant tout basé sur la réussite d'une combinaison d'outils, de raisonnements et de techniques qui se sont avérées nécessaires pour atteindre nos objectifs.

Notre approche à été construite sans laisser de côté la qualité du système, qualité renforcée pas notre générateur de triggers.

Comme montré dans nos diverses illustrations, la rétro-ingénierie des bases de données, même pour un cas en apparence simple, est un processus complexe.
Le chemin est souvent long, difficile, risqué et coûteux et les résultats ne correspondent pas toujours aux attentes.

9.2 Perspectives futures

Des nouvelles voies de recherche doivent encore être explorées dans lesquelles certains de nos outils, analyses ou méthodes proposées pourraient être utilisés moyennant certaines adaptations ou améliorations.

De plus, certaines limitations pourraient être levées moyennant certains ajustements.

Il serait intéressant d'étudier comment nos plugins ou notre méthodologie pourraient être couplés avec des outils plus puissants ou d'autres processus afin de proposer une solution plus exhaustive.

D'un point de vue pratique, une des perspectives futures à envisager et qui semble intéressante à réaliser, serait de se tourner vers un analyseur de pattern exploitant notre catalogue.

La recherche manuelle de ceux-ci s'avère une tâche lourde, coûteuse et hasardeuse. De plus, elle peut comporter des risques d'erreurs. Il est donc nécessaire de développer un outil d'aide permettant de localiser ces patterns.

A cet effet, l'atelier DB-MAN dispose de nombreux outils de matching et d'analyse de code source.

Une autre perspective serait de se tourner vers la recherche de contraintes dans le code procédural à l'aide de méthodes d'analyse dynamique, ce qui viendrait enrichir le processus d'analyse et de surcroît notre catalogue de patterns.

Il faut avouer que la recherche de requêtes SQL statiques est plutôt simple alors que l'implémentation de techniques pour procéder à une analyse dynamique des requêtes SQL est bien plus complexe et fastidieuse.

Certaines recherches relatant du « traçage » du flux circulant vers la base de données montrent que ce processus de recherche dynamique est tout à fait envisageable et réaliste [Cleve 2008].

Même si beaucoup de chemin a déjà été fait dans le domaine de la rétro-ingénierie des bases de données, beaucoup d'autres choses doivent et peuvent encore être explorées.
Le chemin est long et semé d'embûches. De nombreuses questions restent encore sans réponse.

Enfin, les problèmes que nous avons considérés jusqu'alors ne prennent en compte qu'une partie des contraintes existantes. Il serait intéressant d'étoffer nos plugins et nos recherches pour considérer le cas d'autres contraintes.
Une étape supplémentaire serait de tester nos outils sur une application complète du domaine de l'entreprise afin d'évaluer plus concrètement l'efficacité de notre approche.

De plus, notre catalogue manque encore d'expérience. Il a besoin d'être enrichi pour être plus mature et plus complet. En effet, celui-ci est encore loin d'être exhaustif.

Pour renforcer un peu plus la qualité, il serait intéressant de développer un générateur de code garantissant le respect de certaines contraintes, l'ingénieur n'aurait plus qu'à l'intégrer dans le code source.

Beaucoup de recherches restent donc encore à faire dans ce domaine et constituent un travail qui dépasse le cadre de ce mémoire.

Par la variété de l'approche et les nombreux exemples intuitifs proposés dans ce mémoire, nous espérons avoir donné au lecteur un ouvrage instructif qui lui permettra de se faire une meilleure idée sur la compréhension des différentes disciplines que comporte le domaine de la rétro-ingénierie des bases de données.

ACRONYMES

BD	Base de Données.
BNF	Backus-Naur Form.
CARE	Computer-Aided Reverse Engineering.
CASE	Computer-Aided Software Engineering.
COBOL	Common Business Oriented Language.
CODASYL	Conference on Data Systems Languages.
DF	Dépendance Fonctionnelle.
DMS	Data Management System (Système de Gestion de Données).
E/A	Entité Association.
FN	Forme Normale.
FNBC	Forme Normale de Boyce-Codd.
IMS	Information Management System (Système de Gestion de l'Information).
LDD	Langage de Description des Données.
LMD	Langage de Manipulation de Données.
MCD	Modèle Conceptuel de Données.
MEA	Modèle Entité Association.
MYSQL	My Structured Query Language.
OMT	Object Modeling Technique (Modélisation et Conception Orientée Objet).
OO	Orienté Objet.
SGBD	Système de Gestion de Base de Données.
SGBDR	Système de Gestion de Base de Données Relationnelle.
SQL	Structured Query Language (Langage Structuré de Requêtes).

BIBLIOGRAPHIE

[Akoka 1999] AKOKA J., Comyn-Wattiau I. & Lammari N., « *Relational Database ReverseEngineering: Elicitation of Generalization Hierarchies* », Proceedings on workshop REIS'99 de ER'99, LNCS 1727, Paris, 1999.

[Alexander 1964] ALEXANDER C., "*Notes on the Synthesis of Form*". Harvard University Press, 1964.

[Alexander 1977] ALEXANDER C., "*A Pattern Language: Towns, Buildings, Construction*". Oxford University Press, 1977.

[Alexander 1979] ALEXANDER C., "*The Timeless Way of Building*". Oxford University Press, 1979.

[Andersson 1994] ANDERSSON Martin, «*Extracting an Entity Relationship Schema from a Relational Database through Reverse Engineering.*», 1994.

[Anderson 1996] ANDERSSON M, Reverse Engineering of Legacy Systems: From Value-Based to Object-Based Models. PhD thesis, EPFL, Switzerland, 1996.

[Barbar 2002] BARBAR A., « *Extraction de connaissances pour la rétro-conception d'une base de données vers un schéma objet* » thèse soutenue le 28/10/2002, Université de Nice-Sophia Antipolis.

[Batini 1992] BATINI C., Ceri S., Navathe B., « *Conceptual Database Design – An Entity-Relationship Approach* », Benjamin/Cummings, 1992.

[Blaha 1995] BLAHA M., Premerlani J., «*Observed Idiosyncracies of Relational Database Designs*» on Proc. of the 2nd Working Conf. on Reverse Engineering (WCRE'95),IEEE Computer Socity Press.Toronto, 1995.

[Blaha 1998] BLAHA M., Premerlani J., « *Object-Oriented Modeling and Design for Database Applications* », Prentice-Hall, 1998.

[Britton 1989] BRITTON D., Millman J. and Torgerson S. « *A Feasibility and Performance Study of Dependency Inference* », Proc. IEEE Int. Conf. of Data and Engineering (ICDE), 1989.

[Campbell 1994] CAMPBELL L., Halpin T. « *The reverse engineering of relational databases* », in Proc. 6th Int. Work. on CASE, 1994.

[Casanova 1983] CASANOVA Marco A., Amaral de Sa J., « *Designing Entity-Relationship Schemes for Conventional Information Systems* », 1983.

[Casanova 1984] CASANOVA Marco A., Amaral de Sa J., « *Mapping uninterpreted Schemes intoEntity-Relationship diagrams : two applications to conceptual schema design, in IBM J.Res. & Develop.*», 1984.

[Chen 1976] CHEN P., « *The Entity-Relationship Model - Toward a Unified View of Data, ACM Transactions on Database Systems* », vol. 1, no. 1, pp. 9-36, 1976.

BIBLIOGRAPHIE

[Chiang 1993] CHIANG R.H.L., Barron T.M., Storey V.C., « *Performance Evaluation of Reverse Engineering Relational Databases into Extended Entity-Relationship Models* », in Proc. du 12th International Conference on Entity-Relationship Approach, R. Elsmari and V. Kouramajian (Eds.), Arlington, Texas, USA, pp 336-352, Dec 1993.

[Chiang 1994] CHIANG R.H.L., Barron T.M., Storey V.C., « *Reverse Engineering of Relational Databases: Extraction of an EER model from a Relational Database* », Data & Knowledge Engineering Vol n°12, 1994.

[Chiang 1995] CHIANG R.H.L., Barron T.M., Storey V.C., « *A knowledge-based system for performing reverse engineering of relational databases* », Decision Support Systems 13, pp 295-312, North-Holland, 1995.

[Chiang 1996] CHIANG R.H.L., Barron T.M., Storey V.C. « *A framework for the design and evaluation of reverse engineering methods for relational databases* », *Data and Knowledge Engineering*, Vol. 21, No. 1, 1996.

[Chikofsky 1990] CHIKOFSKY E.J., Cross II J.H, « *Reverse engineering and design recovery* » A taxonomy.IEEE Software, 13, 1990.

[Cleve 2008] CLEVE Anthony, Hainaut J-L., « *Dynamic Analysis of SQL Statements for Data-Intensive Applications Reverse Engineering* », proceedings of the 15th Working Conference on Reverse Engineering (WCRE'08), IEEE Computer Society Press, 2008.

[Cleve 2008a] CLEVE Anthony, Hainaut J-L., Lemaitre J., Henrard J., Mouchet C. « *The Role of Implicit Schema Constructs in Data Quality* », 2008.

[Codd 1970] CODD E.F., *"A Relational Model of Data for Large Shared Data Banks"*, Communications ACM, V13, N6, Juin 1970.

[Comyn 1996] COMYN-WATTIAU Isabelle, Akoka Jacky, « *Reverse Engineering of Relational Database Physical Schema.*», 1996.

[Date 1989] DATE C.J. : A Guide to the SQL Standard, 2nd ed. Addison–Wesley Publishing Company, 1989.

[Davis 1985] DAVIS K. H., Adarsh, K.A., «*A Methodology for Translating a Conventional File System into an Entity-Relationship Model, in Proc. of Entity-Relationship Approach*», 1985.

[Davis 1988] DAVIS Kathi Hogshead, Adarsh K. Arora, « *Converting A Relational Database Model into an Entity-Relationship Model* » in Proc. of Entity-Relationship Approach : a Bridge to the User, 1988.

[Davis 2000] DAVIS Kathi Hogshead, « *Data Reverse Engineering: A Historical Survey*», Proceedings of the Seventh Working Conference on Reverse Engineering, Brisbane, Australia, IEEE Computer Society, pages 70-78, Nov. 200.

[DBMAIN] DB-Main project, Computer-aided Database Engineering - Volume 1: Database Models (fourth edition), technical document, Institut d'informatique, FUNDP, 1999. (http://www.db-main.be)

BIBLIOGRAPHIE

[Delobel 1982] DELOBEL C., Adiba M., « *Bases de données et systèmes relationnels* », BORDAS, Paris, 1982.

[Edwards 1995] EDWARDS H., Munro M. «*Deriving a Logical Model for a System Using Recast Method*», in Proc. of the 2nd IEEE WC on Reverse Engineering, Toronto, IEEE Computer Society Press, 1995.

[Fong 1994] FONG J., Ho M.. «*Knowledge-based Approach for Abstracting Hierarchical and Network Schema Semantics*», in Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas, Springer-Verlag, 1994.

[Fong 1997] FONG J., «*Converting Relational to Object-Oriented Databases*». SIGMOD Record, vol 26, n°1, 1997.

[Fonkam 1992] FONKAM M. M., Gray W. A., « *An Approach to Eliciting the Semantics of Relational Databases*» , 1992.

[Gro 1998] GROTENHUIS K., «*Crossing the Euro rubicon*», IEEE Spectrum, 35(10):30–33, 1998.

[Hainaut 1991] HAINAUT J-L., « *Database Reverse Engineering: Models, Techniques, and Strategies* » , 1991.

[Hainaut 1992] HAINAUT J-L., M. Cadelli, B. Decuyper, O. Marchand, « *Database CASE Tool Architecture: Principles for Flexible Design Strategies* », 1992.

[Hainaut 1993a] HAINAUT J-L., Chandelon M., Tonneau C., Joris M. « *Contribution to a Theory of Database Reverse Engineering* », in Proc, 1993.

[Hainaut 1993b] HAINAUT J-L., Tonneau C., Joris M., Chandelon M., « *Schema Transformation Techniques for Database Reverse Engineering* » in Proc. of the 12th Int. Conf. on ER Approach, Arlington-Dallas, E/R Institute and Springer-Verlag, LNCS, 1993.

[Hainaut 1994] HAINAUT J-L., Englebert V., Henrard J., HickJ.-M., Roland D.: «*Evolution of Database Applications: The DB-MAIN Approach.* » In Proc. of the 13th Int. Conf. on ER Approach (ER'94), Manchester, 1994. Springer-Verlag.

[Hainaut 1997a] HAINAUT J-L., Henrard J., Hick J-M., Roland D., Englebert V. « *Contribution to the Reverse Engineering of OO Applications - Methodology and Case Study* », in Proc. of the IFIP 2.6 WC on Database Semantics (DS-7), Leysin (CH), Chapman-Hall, Oct. 1997.

[Hainaut 1997b] HAINAUT J-L., Englebert V., Hick J-M., Henrard J, Roland D., « *Knowledge Transfer in Database Reverse Engineering - A Supporting Case Study* », in Proceedings of the IEEE Working Conference on Reverse Engineering, IEEE Computer Society Press,Amsterdam, Oct. 1997.

[Hainaut 1997c] HAINAUT J-L., Hick J.-M., Henrard J., Englebert V., Roland D. « *The Concept of Foreign key in Reverse Engineering: A Pragmatic Interpretative Taxonomy*». Technical report, Computer Science Departement, University of Namur, Belgium, 1997.

BIBLIOGRAPHIE

[Hainaut 2000] HAINAUT J-L., «*The Nature of Data Reverse Engineering* », *Data Reverse Engineering Workshop*, EuroRef, Seventh Reengineering Forum,Reengineering Week 2000, Zurich, Switzerland, Mars 2000.

[Hainaut 2002] HAINAUT J-L., « *Introduction to Database Reverse Engineering* », 2002.

[Hainaut 2006] HAINAUT J-L., (2006-2007) « Cours d'Ingénierie des Bases de données (Cinquième édition) » Année 2006-2007 http://www.fundp.ac.be/etudes/cours/page_view/IHDCB334/

[Henrard 2000] HENRARD J., Hainaut J-L., Hick J-M., Roland D. and Englebert V., « *From micro-analytical Techniques to Mass Processing in Data Reverse Engineering –The Economic Challenge*», DataReverse Engineering Workshop, EuroRef, Seventh Reengineering Forum, Reengineering, Zurich, Switzerland, Mars 2000.

[Henrard 2002] HENRARD J., Roland D., Englebert V., Hick J-M., Hainaut J-L, « *outils d'analyse de programmes pour la rétro-conception de bases de données* », 2002.

[Henrard 2003] HENRARD J., « *Program Understanding in Databases Reverse Engineering* », thèse présentée en août 2003.

[Hick 2001] HICK Jean-Marc, « *Evolution d'applications de bases de données relationnelles : Méthodes et Outils* », 2001.

[IBM 1998] IBM, « *The Year 2000 and 2-Digit Dates: A Guide for Planning and Implementation. Technical report* » GC28-1251-08, 1998.

[Jahnke 1999] JAHNKE J.-H, «*Management of Uncertainty and Inconsistency in Database Reengineering Processes* ». Ph.D. Dissertation. Paderborn, Germany, Aug. 1999.

[Jahnke 1999b] JAHNKE J.-H, Jörg Wadsack « *Human-Centered Tool Support for Database Reengineering* », 1999.

[Jarzabek 1998] JARZABEK Stan, Huang Riri. « *The case for user-centered case tools* », 1998

[Johannesson 1990] JOHANNESSON P., Kalman, K. « *A method for transforming Relational Schemas into Conceptual Schemas* », dans Proc. of the 8th ERA conference, Toronto, North-Holland, 1990.

[Lammari 1999] LAMMARI N., « *An Algorithm to Extract Is_A Inheritance Hierarchies from a Relational Database* », Proceedings de ER'99, LNCS 1728, Paris, 1999.

[Lammari 2007] LAMMARI Nadira, Comyn-Wattiau Isabelle, Akoka Jacky. «*Extracting generalization hierarchies from relational databases: A reverse engineering approach* », 2007.

[Markosian 1994] MARKOSIAN L., Newcomb P., Brand R., Burson S. and Kitzmiller T., « *Using an enabling technology to reengineer legacy systems* », , 1994.

[Markowitz 1990] MARKOWITZ V., Markowsky J.A., « *Identifying Extended Entity-Relationship Object Structures in Relational Schemas* », IEEE Transactions on Software Engineering, Vol.16, N.8, Août 1990.

BIBLIOGRAPHIE

[Martin 1997] MARTIN R.A., *« Dealing with dates: Solutions for the Year 2000 »*, 1997.

[Milano 2005] MILANO D., Scannapieco M., and Catarci T., *« Using ontologies for xml data cleaning »*. In OTM Workshops,pages 562–571. Springer, 2005.

[Missaoui 1995] MISSAOUI R., Gagnon J. M., Godin R., *«Mapping an Extented Entity-Relationship Schema into a Schema of Complex Objets.»* OOER'95, LNCS 1021, Gold Coast, Australia, 1995.

[Navathe 1988] NAVATHE Shamkant B., Awong A., *«Abstracting Relational and Hierarchical Data with a Semantic Data Model»* dans Proc. of Entity-Relationship Approach : a Bridge to the User, 1988.

[Nilsson 1985] NILSSON Erik G., *«The Translation of a Cobol Data Structure to an Entity-Relationship Type Conceptual Schema »* dans Proc. of Entity-Relationship Approach, October, 1985.

[Pedro 1999] PEDRO-DE-JESUS L., Sousa P., *« Selection of reverse engineering methods for relational databases »*,Software Maintenance and Reengineering,. Proceedings of the Third European Conference on Volume , Issue, Page(s):194 – 197, 1999.

[Petit 1994] PETIT J-M., Kouloumdjian J., Boulicaut J-F., Toumani F., *« Using Queries to Improve Database Reverse Engineering »*, dans Proc. of the 13th International Conference on Entity-Relationship Approach, Manchester, UK, Dec. 1994.

[Petit 1995] PETIT, J-M., Kouloumdjian J., Toumani F., *« Relational Database Reverse Engineering: a Method Based on Query Analysis »*., 1995.

[Petit 1996a] PETIT, J-M., Kouloumdjian J., Boulicaut J-F., Toumani F., *« Towards the Reverse Engineering of of Denormalized Relational Databases »*, dans Proc. of the 12th International Conference on Data Engineering, New Orleans, Louisiana, USA, IEEE Press, Fev.1996.

[Petit 1996b] PETIT J-M. *«Fondements pour un Processus Réaliste de Rétro-Conception de Bases de Données Relationnelles»*. PhD thesis, University Lyon I, France, 1996.

[Premerlani 1994] PREMERLANI W.J., Blaha M.R., *« An Approach for Reverse Engineering of Relational Databases »*, Communications pf the ACM, vol. 37, n.5, 1994.

[Rama 1997] RAMANATHAN S. et Hodges J., *«Extraction of Object-Oriented Structures from Existing Relational Databases»*, SIGMOD Record, vol 26, n°1, 1997.

[Sabanis 1992] SABANIS N., Stevenson *« N. Tools and Techniques for Data Remodelling Cobol Applications»*, in Proc. 5th Int. Conf. on Software Engineering and Applications, Toulouse, 7-11, pp. 517-529, 1992.

[Shoval 1993] SHOVAL P., Shreiber N., *« Database Reverse Engineering : from Relational to the Binary Relationship Model, Data and Knowledge Engineering»*, Vol. 10, No. 10, 1993.

[Signore 1994] SIGNORE O., Loffredo M., Gregori M., Cima M., *« Using Procedural Patterns in Abstracting Relational Schemata »*, in Proc. of the 13th International Conference on Entity-Relationship Approach, 1994.

BIBLIOGRAPHIE

[Signore 1994b] SIGNORE O, Loffredo M., Gregori M., Cima M., « *Reconstruction of ER Schema from Database Applications: a Cognitive Approach* », in Proc. of the 13th Int. Conf. on ER Approach, Manchester, Springer-Verlag, 1994.

[Spring 1990] SPRINGSTEEL F., Kou C., «*Reverse Data Engineering of E-R designed Relational schemas*», Parallel Architectures and their Applications, 1990.

[Sql 1992] Information Technology - Database Language SQL (Proposed revised text of DIS 9075) (Second Informal Review Draft) ISO/IEC 9075, July, 1992.

[Tardieu 1983] TARDIEU H., Rochfeld, Coletti, La méthode Merise, Les Editions d'Organisation, 1983.

[Tari 1998] TARI Zahir, Omran Bukhres, Stokes John, Hammoudi Slimane «*The Reengineering of Relational Databases based on Key and Data Correlations*», 1998.

[Teo 1986] TEOREY T.J., Yang D., «*Fry J.P. A Logical Design Methodology for Relational Databases Using the Extended Entity-Relationship Model*», ACM Computing Surveys, Vol.18, No.2, Jun. 1986.

[Teo 1989] TEOREY T.J., «*Database Design* », 1989.

[Teo 1990] TEOREY T.J., «*Database Modelling and Design - The Entity-Relationship Approach* », 1990.

[Theodoros 1998] THEODOROS L., Edwards H., Bryant A., «*Reverse Engineering from OO Source Code to OMT Design*», in Proc. of the 5th IEEE Working Conf. on Reverse Engineering, Honolulu, October 1998, IEEE Computer Society Press,1998.

[Vermeer 1995] VERMEER Mark W. W, Apers Peter M. G, «Reverse Engineering of Relational Database Applications.», 1995.

[Winans 1990] WINANS John, Davis Kathi Hogshead «*Software Reverse Engineering from a Currently Existing IMS Database to an Entity-Relationship Model* », 1990.

[White 2004] WHITE Simon, « *How to Strike a Match* » , 2004.

[Wikipedia] Encyclopédie multilingue, universelle, librement diffusable, disponible sur le web. Créée en janvier 2001. (www.wikipedia.org)

Facultés Universitaires Notre-Dame de la Paix, Namur

Institut d'Informatique

Année académique 2008 - 2009

Co-analyse Schéma-Données-Programmes en rétro-ingénierie des bases de données

ANNEXES

Folisi Piero

Mémoire présenté en vue de l'obtention du grade de
Licencié en Informatique

ANNEXE A : Présentation des plugins

A.1 Introduction

La rétro-ingénierie réclame sans cesse de nouveaux raisonnements, de nouvelles techniques et outils. Nous avons d'ailleurs très vite compris qu'il nous était impossible de mener à bien l'objectif de ce mémoire sans leur utilisation.

En effet, devant la grande variété et la grande quantité d'informations à analyser (code source, données, schéma, etc.) les tâches de recherche deviennent de plus en plus lourdes et complexes. L'ingénieur du domaine de la rétro-ingénierie a donc besoin d'outils pour accomplir efficacement son travail.

Il nous semblait donc intéressant de développer des plugins spécifiques à l'aide de l'API Java de DB-MAIN (JIDBM). Nous présenterons dans cette annexe les différents plugins mettant en œuvre notre méthodologie de travail.

A.1.1 DB-MAIN

Le projet de recherche DB-MAIN a démarré en septembre 1993 à l'Université de Namur. DB-MAIN est un environnement de génie logiciel (atelier de génie logiciel ou AGL) destiné à l'ingénierie des bases de données. Il a pour objectif d'apporter une aide dans les principaux processus de développement et de maintenance de bases de données.
Son but est d'aider l'ingénieur aussi bien dans le développement, dans l'évolution, dans la migration ou dans l'intégration de la base de données. Il est également utilisé dans la réalisation des processus de rétro-ingénierie; on parle alors d'outils CARE[59]. Celui-ci inclut également de nombreux outils d'aide au développement ou à la rétro-conception des bases de données.

De plus amples informations peuvent être trouvées dans [Hainaut 1994] et [DBMAIN]. Un aperçu des fonctionnalités de l'atelier ainsi qu'un tutoriel sont disponibles à l'adresse http://www.dbmain.be/references.html.

[59] Computer-Aided Reverse Engineering.

A.1.2 JIDBM

La librairie JIDBM[60] (Java Interface for DB-MAIN) est une API java permettant à l'utilisateur de développer ses propres modules JAVA accédant au référentiel et aux fonctions de DB-MAIN.

L'architecture de cette API est illustrée dans la figure A-1. Elle fonctionne avec l'aide de deux librairies jidbm.dll et jidbm.jar :

- jidbm.dll : permet à JIDBM de tourner à l'intérieur de la machine virtuelle JAVA pour fonctionner avec reposit.dll (la librairie principale du repository de DB-MAIN)
- jidbm.jar : package de JIDBM contenant les classes permettant aux programmeurs d'accéder au repository de DB-MAIN.

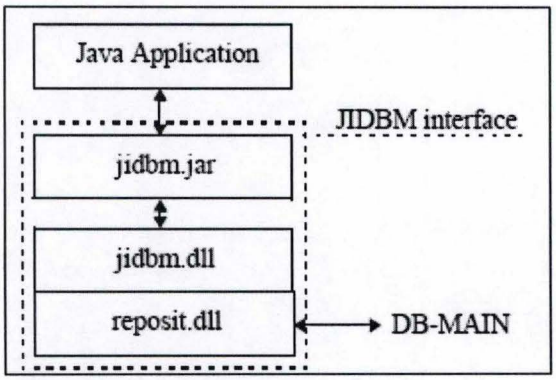


Figure A-1 Architecture de l'API de DB-MAIN.

A.2 Présentation des plugins

Cette section a pour objet d'expliciter brièvement nos plugins (interfaces, rapports et fichiers générés) et de donner un bref descriptif et quelques chiffres sur l'implémentation de ceux-ci.

A.2.1 Présentation générale

L'implémentation des différents composants a été réalisée en Java. Pour la gestion de la base de données nous avons utilisé le SGBD MySQL. Une base de données a été nécessaire en vue de stocker les informations relatives pour nos dictionnaires de synonymes et de traduction (figures A-3 et A-4).

[60] Le manuel de référence JIDBM est disponible sur http://www.rever.eu/DISTRIBUTION/DB-MAIN/JIDBM-reference-manual.pdf.

ANNEXE A : Présentation des plugins

L'application est composée d'environs 5000 lignes de code comprenant 4 classes principales et 9 classes utilitaires (figure A-5).

La figure A-2 donne une illustration de son fonctionnement. Celle-ci analyse en entrée un schéma défini dans l'environnement DB-MAIN (fichier lun). Suite aux résultats obtenus l'utilisateur aura la possibilité de générer divers rapports (PDF) ou scripts SQL.

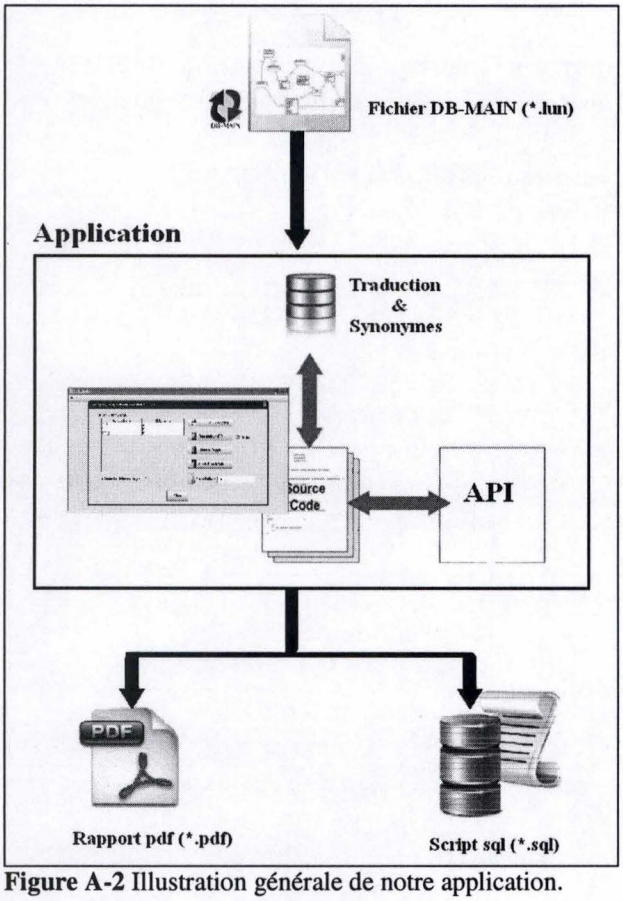


Figure A-2 Illustration générale de notre application.

Les figures A-3 et A-4 donnent une description du contenu des tables traduction et synonyme et du code LDD ayant servis à leur création. Ces tables ont été créées dans un but illustratif. Leurs implémentations dans le SGBD ne constituent pas une solution optimale pour la gestion d'un dictionnaire de traduction ou de synonymes.

	ID_TRANSLATION *	WORD *	TRANSLATIONWORD *
▸	1	OUVRAGE	WORK;BOOK;BOEK;WERK;
	2	WORK	TRAVAIL;WERK
	4	BOEK	LIVRE;OUVRAGE;BOOK
	5	BOOK	BOEK;OUVRAGE

	ID_SYN *	WORD *	SYNONYMWORD *
▸	1	OUVRAGE	TRAVAIL;
	2	TRAVAIL	OUVRAGE;
	5	CUSTOMER	CLIENT

Figures A-3 Contenu des tables synonyme et traduction.

```
CREATE TABLE  SYNONYM  (
   ID_SYN  int(10) NOT NULL AUTO_INCREMENT,
   WORD  char(200) NOT NULL,
   SYNONYMWORD  char(200) NOT NULL,
  PRIMARY KEY ( ID_SYN )
)

CREATE TABLE  TRANSLATION  (
   ID_TRANSLATION  int(10) NOT NULL AUTO_INCREMENT,
   WORD  char(200) NOT NULL,
   TRANSLATIONWORD  char(200) NOT NULL,
  PRIMARY KEY ( ID_TRANSLATION )
)
```

Figure A-4 Code LDD des tables synonyme et traduction.

La figure A-5 donne une brève description du contenu des packages de l'application.

Package	Classe	Description
plugins.assistant	MainFrame	Interface graphique de l'application. Contient la méthode Main de l'application. Celle-ci a besoin de paramètres en entrée pour la connexion au serveur MySQL : <hôte> <port> <nom de la bd> <utilisateur><mot de passe>
	Schema	Classe de gestion des objets de type Schema.
	Entite	Classe de gestion des objets de type Entite.
	Attribut	Classe de gestion des objets de type Attribut.
pdf	Utilz	Classe de gestion pour la création des fichiers pdf.
msgbox	MessageBoxes	Classe de gestion pour les messages box.
filter.jtextfield	IntegerCaseDocument	Classe permettant de rentrer que des chiffres dans un Jtextfield.
	IntLenghMax	Classe permettant de rentrer un nombre limité de chiffres dans un Jtextfield.
	JIntegerField	Classe permettant de faire un mask sur les Jtextfield en ne prenant en compte que l'insertion de chiffres.
	TextLenghMax	Classe permettant de rentrer un nombre limité de caractères dans un Jtextfield.
filechooser	MyFileChooser	Classe de gestion du FileChooser pour l'ouverture des fichiers DB-MAIN.
	ExampleFileFilter	Classe de filtrage du type des fichiers (*.lun).
database	ConnectionDB	Classe de gestion pour la base de données.
plugins.assistant.pictures		Contient les images des différentes icônes utilisées.

Figure A-5 Packages de l'application.

ANNEXE A : Présentation des plugins

Les fichiers générés par l'application respecteront la nomenclature définie dans la figure A-6.

Plugin	Nomenclature du ficher output	Description
Analyseur de schéma	Schema_Analysis_PK_<nom du schéma analysé>.pdf	Fichier PDF du résultat de l'analyse des clés primaires par le schéma.
	Schema_Analysis_FK_<nom du schéma analysé>.pdf	Fichier PDF du résultat de l'analyse des clés étrangères par le schéma.
Analyseur de données	Data_Analysis_PK_<nom du schéma analysé>.pdf	Fichier PDF du résultat de l'analyse des clés primaires par les données suite à l'analyse du schéma.
	Data_Analysis_FK_<nom du schéma analysé>.pdf	Fichier PDF du résultat de l'analyse des clés étrangères par les données suite à l'analyse du schéma.
	SQL_Data_Analysis_PK_<nom du schéma analysé>.sql	Fichier SQL contenant les requêtes générées pour la production du fichier PDF de l'analyse des données sur les clés primaires.
	SQL_Data_Analysis_FK_<nom du schéma analysé>.sql	Fichier SQL contenant les requêtes générées pour la production du fichier PDF de l'analyse des données sur les clés étrangères.
	SQL_Check_Data_Error_PK_<nom du schéma analysé>.sql	Fichier SQL contenant les requêtes de vérification générées pour détecter les enregistrements violant une contrainte d'unicité et d'existence.
	SQL_Check_Data_Error_FK_<nom du schéma analysé>.sql	Fichier SQL contenant les requêtes de vérification générées pour détecter les enregistrements violant une contrainte d'intégrité référentielle.
Générateur de triggers	Trigger_PK_<nom du schéma analysé>. sql	Fichier SQL contenant les triggers générés par rapport aux clés primaires identifiées suite à l'analyse du schéma.
	Trigger_FK_<nom du schéma analysé>. sql	Fichier SQL contenant les triggers générés par rapport aux clés étrangères identifiées suite à l'analyse du schéma.

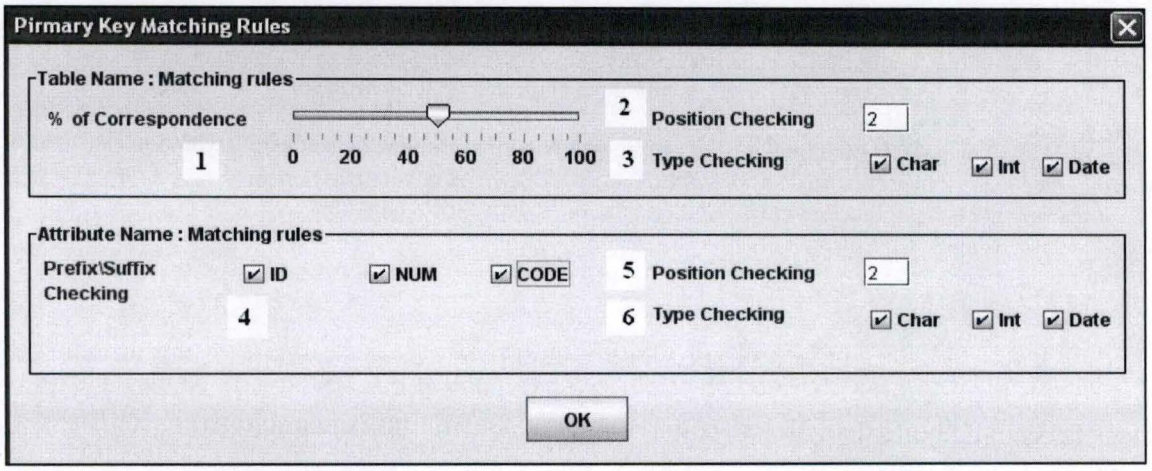
Figure A-6 Nomenclature des fichiers outputs de l'application.

A.2.2 Analyseur de schéma

Les contraintes référentielles et les identifiants sont les constructions les plus importantes à éliciter dans la plupart des projets de rétro-ingénierie de bases de données. C'est pourquoi un analyseur a été implémenté pour découvrir ces constructions potentielles à travers le schéma. Les heuristiques de cet analyseur ont été évoquées dans le chapitre 4.

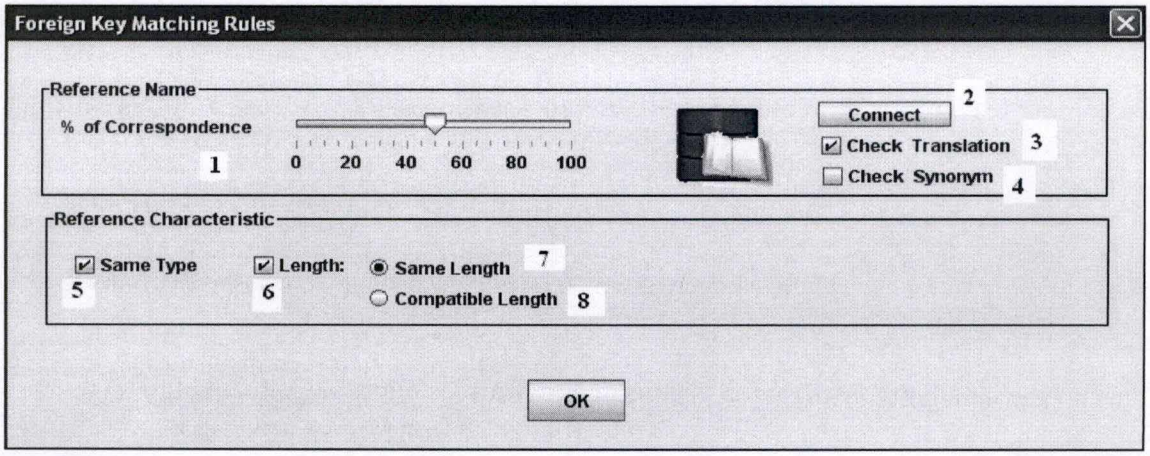
Le plugin pour la recherche des clés primaires ou des clés étrangères est divisé en deux boîtes de dialogue.

Dans la première boîte de dialogue, l'utilisateur définit les règles de « matching » et donc les heuristiques à appliquer pour la recherche (figures A-7 et A-8).



1. Définit le pourcentage de ressemblance entre le nom de l'attribut et son entité.
2. Si complété, le champ définit le nombre de positions maximum devant être vérifiées dans la table pour la ressemblance. Si celui-ci n'est pas complété toutes les positions seront prises en compte.
3. Si une case est cochée (au moins une doit-être cochée), celle-ci définit le(s) type(s) de l'attribut devant être analysé pour la ressemblance.
4. Si une case est cochée, celle-ci définit que le nom de l'attribut peut contenir les mots clés « ID », « NUM » ou « CODE » dans son préfixe ou son suffixe.
5. Si complété, le champ définit le nombre de positions maximum devant être vérifiées dans la table pour la vérification des mots clés.
6. Si une case est cochée (au moins une doit-être cochée), celle-ci définit le(s) type(s) de l'attribut devant être analysé pour la vérification des mots clés.

Figure A-7 Interface servant au matching des clés primaires.



1. Définit le pourcentage de ressemblance entre le nom de l'attribut enfant et celui de son parent.
2. Permet d'établir la connexion avec la base de données contenant les tables synonymes et traduction.
3. Si la case est cochée, le nom de l'attribut enfant peut être une traduction de son parent. Celle-ci est définie grâce à notre dictionnaire de traduction.
4. Si la case est cochée, le nom de l'attribut enfant peut être un synonyme de son parent. Celui-ci est défini grâce à notre dictionnaire de synonymes.
5. Si la case est cochée, l'attribut enfant doit être du même type que son parent.
6. Si la case est cochée, l'attribut enfant doit avoir la même taille ou une taille compatible avec son parent.
7. Critère de taille identique.
8. Critère de taille compatible.

Figure A-8 Interface servant au matching des clés étrangères.

Dans la seconde boîte de dialogue (figures A-9 et A-10), l'utilisateur choisit le schéma à analyser et par la suite de soit :

- Générer un rapport au format PDF contenant le résultat de l'analyseur de schéma (figure A-11)
- Lancer l'analyseur de données
- Lancer le générateur de triggers
- Générer un script SQL permettant de détecter les données invalides par rapport aux contraintes détectées par l'analyse du schéma.

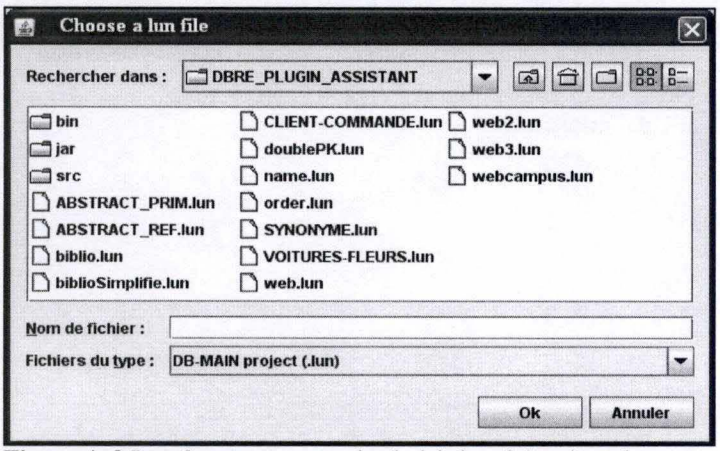
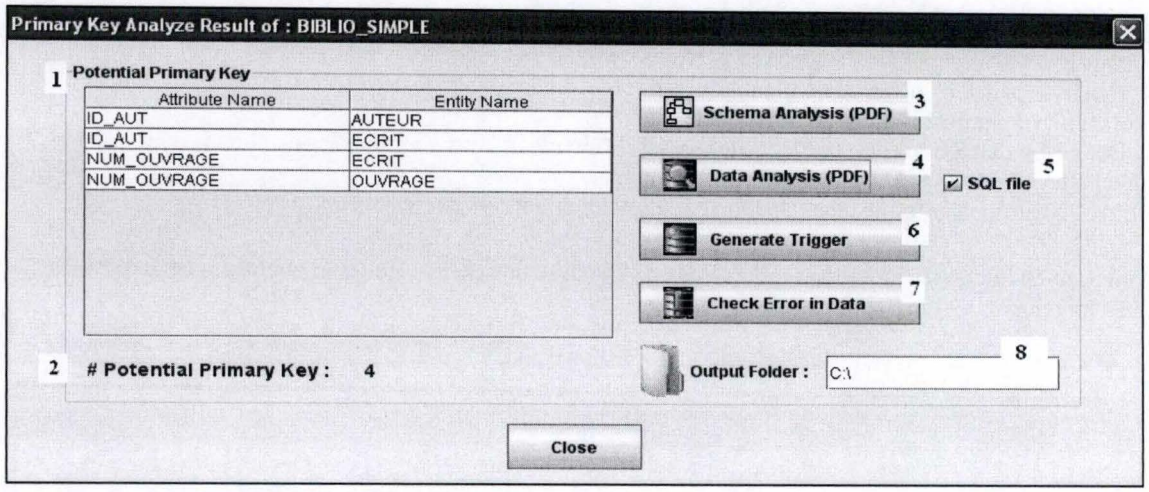


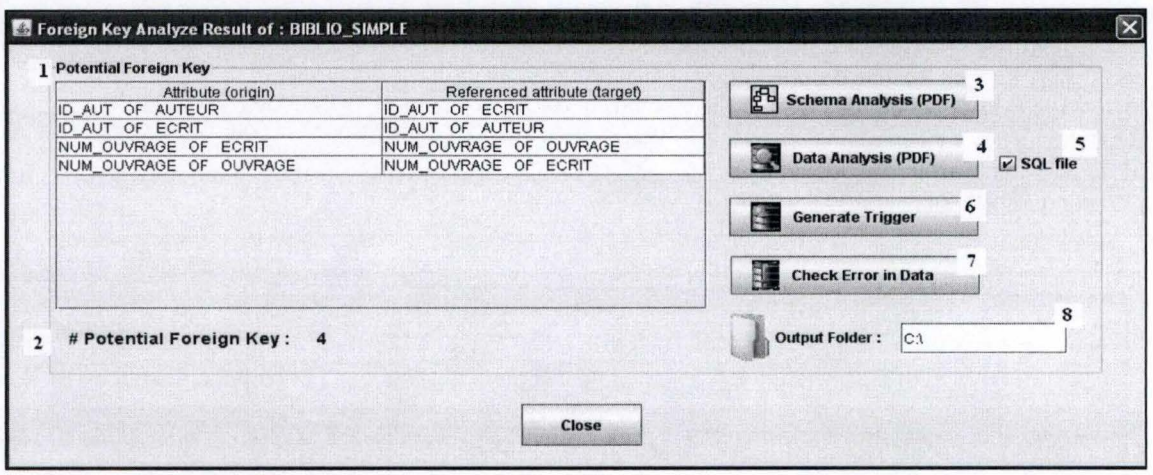
Figure A-9 Interface permettant de choisir le schéma à analyser.

ANNEXE A : Présentation des plugins



1. Clé primaires potentielles ayant été élicitées par l'analyseur de schéma.
2. Nombre de clés primaires potentielles ayant été élicitées par l'analyseur de schéma.
3. Génération d'un rapport PDF contenant le résultat de l'analyse des clés primaires à partir du schéma.
4. Génération d'un rapport PDF contenant le résultat de l'analyse des données à partir des clés primaires ayant été élicitées suite à l'analyse du schéma.
5. Génération du script SQL contenant les requêtes générées pour la production du fichier PDF de l'analyse des données sur les clés primaires.
6. Génération des triggers (script SQL) à partir des clés primaires ayant été identifiées suite à l'analyse du schéma.
7. Génération des requêtes de vérifications (script SQL) permettant de détecter les données violant une contrainte d'unicité et d'existence.
8. Définition du répertoire de sortie contenant les différents fichiers générés par nos plugins pour les clés primaires

a) Interface présentant les résultats de l'analyse des clés primaires par le schéma.



1. Clé étrangères potentielles ayant été élicitées par l'analyseur de schéma.
2. Nombre de clés étrangères potentielles ayant été élicitées par l'analyseur de schéma.
3. Génération d'un rapport PDF contenant le résultat de l'analyse des clés étrangères à partir du schéma.
4. Génération d'un rapport PDF contenant le résultat de l'analyse des données à partir des clés étrangères ayant été élicitées suite à l'analyse du schéma.
5. Génération du script SQL contenant les requêtes générées pour la production du fichier PDF de l'analyse des données sur les clés étrangères.

6. Génération des triggers (script SQL) à partir des clés étrangères ayant été identifiées suite à l'analyse du schéma.
7. Génération des requêtes de vérifications (script SQL) permettant de détecter les éventuelles données violant une contrainte d'intégrité référentielle.
8. Définition du répertoire de sortie contenant les différents fichiers générés par nos plugins pour les clés étrangères.

b) Interface présentant les résultats de l'analyse des clés étrangères par le schéma.

Figures A-10 Interfaces présentant les résultats de l'analyse du schéma. Celles-ci englobent également les autres fonctionnalités de l'application.

DBRE - Schema Analysis - Result Primary Key for : BIBLIO_SIMPLE

N°	Attribute Name	Table Name	Corresp. with Table Name	Pos	Type	Prefix Suffix	In data ?	In pattern ?	Primary key ?
1	ID_AUT	AUTEUR	40,0%	1	INT	ID	yes/no	yes/no	yes/no
2	ID_AUT	ECRIT	0,0%	1	INT	ID	yes/no	yes/no	yes/no
3	NUM_OUVRAGE	ECRIT	0,0%	2	INT	NUM	yes/no	yes/no	yes/no
4	NUM_OUVRAGE	OUVRAGE	75,0%	1	INT	NUM	yes/no	yes/no	yes/no

1. N° : Numérotation des attributs potentiellement identifiés comme clé primaire.
2. Attribute Name : Nom de l'attribut.
3. Table Name : Nom de l'entité contenant l'attribut.
4. Corresp. with Table Name : Pourcentage de ressemblance entre le nom de l'attribut et celui de son entité.
5. Pos : Position de l'attribut dans son entité.
6. Type : Type de l'attribut.
7. Prefix/Suffix : Préfixe ou suffixe identifié dans l'attribut.
8. In data ? : Texte statique destiné à être complété suite à l'analyse des données.
9. In pattern ? : Texte statique destiné à être complété par l'ingénieur suite à l'analyse des patterns.
10. Primary key ? : Texte statique destiné à être complété par l'ingénieur suite à la confrontation des différents composants.

a) Rapport PDF contenant le résultat de l'analyse des clés primaires à partir du schéma.

DBRE - Schema Analysis - Result Foreign Key for : BIBLIO_SIMPLE

N°	Attribute	Referenced Attribute	Corresp with Ref	Trans.	Syn.	Same Type	Length	In data ?	In pattern ?	Foreign key ?
1	ID_AUT (AUTEUR)	ID_AUT (ECRIT)	100,0%	No	No	Yes	Same	yes/no	yes/no	yes/no
2	ID_AUT (ECRIT)	ID_AUT (AUTEUR)	100,0%	No	No	Yes	Same	yes/no	yes/no	yes/no
3	NUM_OUVRAGE (ECRIT)	NUM_OUVRAGE (OUVRAGE)	100,0%	No	No	Yes	Same	yes/no	yes/no	yes/no
4	NUM_OUVRAGE (OUVRAGE)	NUM_OUVRAGE (ECRIT)	100,0%	No	No	Yes	Same	yes/no	yes/no	yes/no

1. N° : Numérotation des attributs potentiellement identifiés comme clé étrangère.
2. Attribute : Nom de l'attribut enfant avec entre parenthèses le nom de son entité.
3. Referenced Attribute : Nom de l'attribut parent avec entre parenthèses le nom de son entité.
4. Corresp. with Ref : Pourcentage de ressemblance entre le nom de l'attribut enfant et celui de son parent.
5. Trans. : Nom de l'attribut enfant est une traduction du nom de l'attribut de son parent.
6. Syn. : Nom de l'attribut enfant est un synonyme du nom de l'attribut de son parent.
7. Same Type : Type identique entre l'attribut enfant et son parent
8. Length : Taille identique ou compatible entre l'attribut enfant et de son parent
9. In data ? : Texte statique destiné à être complété suite à l'analyse des données.
10. In pattern ? : Texte statique destiné à être complété par l'ingénieur suite à l'analyse des patterns.
11. Foreign key ? : Texte statique destiné à être complété par l'ingénieur suite à la confrontation des différents composants

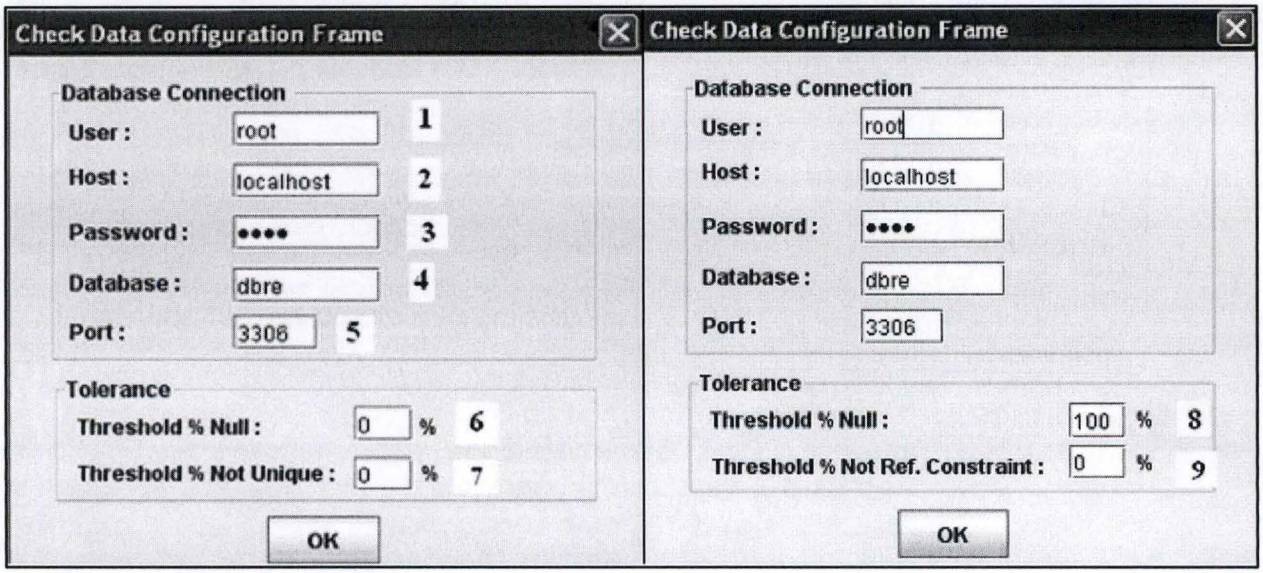
b) Rapport PDF contenant le résultat de l'analyse des clés étrangères à partir du schéma.

Figures A-11 Exemple de rapports PDF générés par l'analyse du schéma.

A.2.3 Analyseur de données

L'analyseur de données est exécuté par la commande « Data Analysis (PDF) » située dans l'interface d'analyse du schéma. Les heuristiques de cet analyseur ont été évoquées dans le chapitre 5 de ce mémoire.

Pour générer le fichier PDF résultant de l'analyse des données l'application a besoin d'établir une connexion avec la base de données contenant les données et les tables à analyser.
Pour ce faire l'utilisateur doit donc encoder les informations relatives dans l'interface de connexion et définir deux seuils de tolérance pour faire face aux éventuelles erreurs pouvant exister dans les données (figure A-12).



1. Utilisateur (paramètre de connexion).
2. Adresse hôte du serveur de la base (paramètre de connexion).
3. Mot de passe (paramètre de connexion).
4. Nom de la base de données (paramètre de connexion).
5. Numéro de port (paramètre de connexion).
6. Seuil de tolérance aux valeurs nulles (contrainte d'existence) pour la vérification des clés primaires.
7. Seuil de tolérance aux valeurs non uniques (contrainte d'unicité) pour la vérification des clés primaires.
8. Seuil de tolérance aux valeurs nulles pour la vérification des clés étrangères.
9. Seuil de tolérance aux valeurs non référencées pour la vérification des clés étrangères.

Figures A-12 Interface de configuration de l'analyseur de données.

La figure A-13 décrit les rapports PDF générés par l'analyseur de données pour la vérification des clés primaires et étrangères.

DBRE - Data Analysis - Result Primary Key for : BIBLIO_SIMPLE

N°	Attribute Name	Table Name	Total Records	% Null	% Not Unique	Threshold %Null - %Not Unique	In data ?	In pattern ?	Primary key ?
1	ID_AUT	AUTEUR	40	0%	0%	0% - 0%	yes	yes/no	yes/no
2	ID_AUT	ECRIT	40	0%	50%	0% - 0%	no	yes/no	yes/no
3	NUM_OUVRAGE	ECRIT	40	0%	50%	0% - 0%	no	yes/no	yes/no
4	ID_AUT,NUM_OUVRAGE	ECRIT	40	0%	0%	0% - 0%	yes	yes/no	yes/no
5	NUM_OUVRAGE	OUVRAGE	40	0%	0%	0% - 0%	yes	yes/no	yes/no

1. N° : Numérotation des attributs potentiellement identifiés comme clé primaire.
2. Attribute Name : Nom de l'attribut.
3. Table Name : Nom de l'entité contenant l'attribut.
4. Total Records : Nombre d'enregistrements dans la table de l'attribut.
5. % Null : Pourcentage de valeurs nulles identifié pour l'attribut.
6. % Not Unique : Pourcentage de doublons identifié pour l'attribut. Attention les valeurs nulles ne sont pas prises en compte dans le calcul du pourcentage.
7. Threshold : Seuil de tolérance aux valeurs nulles et aux doublons. Celui-ci est défini par l'ingénieur.
8. In data ? : Valeur complétée dynamiquement suite à l'analyse des données. Elle est mise à « yes » si le pourcentage de valeurs nulles et le pourcentage de doublons est inférieur ou égal aux seuils de tolérance.
9. In pattern ? : Texte statique destiné à être complété par l'ingénieur suite à l'analyse des patterns.
10. Primary key ? : Texte statique destiné à être complété par l'ingénieur suite à la confrontation des différents composants.

a) Rapport PDF contenant le résultat de l'analyse des données sur les clés primaires identifiées par l'analyse du schéma.

DBRE - Data Analysis - Result Foreign Key for : BIBLIO_SIMPLE									
N°	Attribute	Referenced Attribute	Total Records	% Null	% Not Ref. Constraint	Threshold %Null - %Not Ref.	In data ?	In pattern ?	Foreign Key ?
1	ID_AUT (AUTEUR)	ID_AUT (ECRIT)	10	0%	25%	100% - 0%	no	yes/no	yes/no
2	ID_AUT (ECRIT)	ID_AUT (AUTEUR)	10	0%	0%	100% - 0%	yes	yes/no	yes/no
3	NUM_OUVRAGE (ECRIT)	NUM_OUVRAGE (OUVRAGE)	10	0%	0%	100% - 0%	yes	yes/no	yes/no
4	NUM_OUVRAGE (OUVRAGE)	NUM_OUVRAGE (ECRIT)	10	0%	25%	100% - 0%	no	yes/no	yes/no

1. N° : Numérotation des attributs potentiellement identifiés comme clé étrangère.
2. Attribute: Nom de l'attribut enfant avec entre parenthèses le nom de son entité.
3. Referenced Attribute : Nom de l'attribut parent avec entre parenthèses le nom de son entité.
4. Total Records : Nombre d'enregistrements dans la table de l'attribut enfant.
5. % Null : Pourcentage de valeurs nulles identifié pour l'attribut enfant.
6. % Not Ref Cosntraint : Pourcentage de valeurs de l'attribut enfant ne respectant pas la contrainte d'intégrité référentielle. Attention les valeurs nulles ne sont pas prises en compte dans le calcul du pourcentage.
7. Threshold : Seuil de tolérance aux valeurs nulles et au non respect de la contrainte d'intégrité référentielle. Celui-ci est défini par l'ingénieur.
8. In data ? : Valeur complétée dynamiquement suite à l'analyse des données. Elle est mise à « yes » si le pourcentage de valeurs nulles et le pourcentage des valeurs ne respectant pas la contrainte d'intégrité est inférieur ou égal aux seuils de tolérance.
9. In pattern ? : Texte statique destiné à être complété par l'ingénieur suite à l'analyse des patterns.
10. Primary key ? : Texte statique destiné à être complété par l'ingénieur suite à la confrontation des différents composants.

b) Rapport PDF contenant le résultat de l'analyse des données sur les clés étrangères identifiées par l'analyse du schéma.

Figures A-13 Exemple de rapports PDF générés par l'analyse des données.

A.3.4 Générateur de triggers

Les triggers ou déclencheurs désignent une fonction qui doit s'exécuter avant ou après un événement comme par exemple une commande INSERT, UPDATE ou DELETE.

Le but de notre générateur est de générer des triggers permettant de garantir que les contraintes sur les clés primaires et sur les clés étrangères soient respectées.

Les triggers générés ont été testés dans l'environnement MySQL. La syntaxe de ceux-ci peut donc légèrement différer d'un SGBD à l'autre.

La figure A-14 donnent un exemple des différents triggers générés par notre générateur.

Clé primaire :

Triggers garantissant la contrainte d'unicité de la clé primaire :

```
delimiter //
create trigger CHK_PK_INS_<identifiant>
BEFORE INSERT ON <table>
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM <table> WHERE <identifiant>) = new.<identifiant>) THEN
     call my_error_pk_violation();
END IF;
END//
delimiter ;

delimiter //
create trigger CHK_PK_UPD_<identifiant>
BEFORE UPDATE ON <table>
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM <table> WHERE <identifiant>) = new.<identifiant>) THEN
  IF ((old.<identifiant> <> new.<identifiant>)) THEN
     call my_error_pk_violation();
  END IF;
END IF;
END//
delimiter ;
```

Clé étrangère :

Triggers assurant la gestion de l'intégrité référentielle au niveau de la table enfant :

```
delimiter //
create trigger CHK_FK_CHILD_IN_<ref>_<table enfant>
BEFORE INSERT ON <table enfant>
FOR EACH ROW
BEGIN
IF NOT (new.<ref> is null) THEN
 IF NOT EXISTS (SELECT <identifiant> FROM <table parent> WHERE <identifiant>= new.<ref>) THEN
     call my_error_fk_violation();
 END IF;
END IF;
END//
delimiter ;

delimiter //
create trigger CHK_FK_CHILD_UP_<ref>_<table enfant>
BEFORE UPDATE ON <table enfant>
FOR EACH ROW
BEGIN
IF NOT (new.<ref> is null) THEN
 IF (old.<ref> <> new.<ref>) THEN
  IF NOT EXISTS (SELECT <identifiant> FROM <table parent> WHERE <identifiant>= new.<ref>) THEN
     call my_error_fk_violation();
  END IF;
 END IF;
END IF;
END//
delimiter ;
```

Clé étrangère :

Triggers assurant la gestion de l'intégrité référentielle au niveau de la table parent :

```
delimiter //
create trigger CHK_FK_PARENT_DEL_<identifiant>_<table parent>
BEFORE DELETE ON <table parent>
FOR EACH ROW
BEGIN
 IF EXISTS (SELECT <ref> FROM <table enfant> WHERE <ref>= old.<identifiant>) THEN
     call my_error_fk_violation();
 END IF;
END//
delimiter ;

delimiter //
create trigger CHK_FK_PARENT_UPD_<identifiant>_<table parent>
BEFORE UPDATE ON <table parent>
FOR EACH ROW
BEGIN
IF (old.<identifiant> <> new.<identifiant>) THEN
  IF EXISTS (SELECT <ref> FROM <table enfant> WHERE <ref>= old.<identifiant>) THEN
     call my_error_fk_violation();
  END IF;
END IF;
END//
delimiter ;
```

NB : Pour la génération des triggers au niveau de la table parent nous avons pris l'option la plus saine qui est de générer un blocage en cas de mise à jour ou de suppression de la référence d'une ligne parent si celle-ci est référencée dans une ligne enfant

Figure A-14 Triggers générés par notre générateur de triggers.

ANNEXE B : Illustration des plugins

B.1 Introduction

Cette annexe décrit une petite illustration dans laquelle nos plugins ont été appliqués. Ceci permettra de visualiser la mise en œuvre de notre stratégie d'approche et de donner un aperçu des outils développés. L'objectif majeur de cette illustration est de montrer la faisabilité de notre méthode ainsi que l'efficacité des plugins mis à disposition de l'ingénieur.

Cette illustration a été réalisée sur un exemple académique à savoir « la gestion des clients et des commandes ». Le schéma physique de cette illustration est présenté dans la figure B-1. Celui-ci contient 3 clés étrangères implicites et 4 clés primaires.

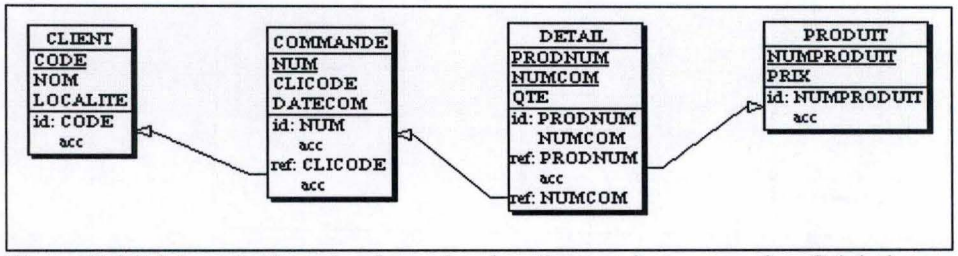


Figure B-1 Schéma physique de «la gestion des clients et des commande». Celui-ci comprend 4 clés primaires et 2 clés étrangères implicites.

A cet effet les quatre tables de l'exemple ont été crées et alimentées avec des données fictives respectant les contraintes, quelques données erronées ont été introduites dans la base.

Pour illustrer au mieux nos plugins, aucune de ces contraintes ne sera codée dans le code LDD (figure B-2). Mis à part les contraintes d'existence, les autres contraintes ne sont donc pas connues du SGBD. Nous supposerons que ce sont les utilisateurs qui gèrent manuellement chaque mise à jour de la base de données. Ceux-ci connaissent donc implicitement les identifiants et les liens logiques entre les tables.

```
CREATE TABLE CLIENT (
   CODE char(6) NOT NULL,
   NOM char(20) NOT NULL,
   LOCALITE char(40) NOT NULL
) ;

CREATE TABLE COMMANDE (
   NUM char(6) NOT NULL,
   CLICODE char(6) NOT NULL,
   DATECOM date NOT NULL
) ;

CREATE TABLE DETAIL (
   PRODNUM char(6) NOT NULL,
   NUMCOM char(6) NOT NULL,
   QTE int(6) NOT NULL
) ;

CREATE TABLE PRODUIT (
   NUMPRODUIT char(6) NOT NULL,
   PRIX decimal(6,2) NOT NULL
) ;
```

Figure B-2 Code LDD des tables de la figure B-3.

Le but est de voir si nos plugins peuvent, via l'analyse du schéma et des données, retrouver les clés primaires et les clés étrangères du schéma de la figure B-1.

B.2 Illustration

B.2.1 Analyse du schéma

Cette activité démarre avec le schéma physique de la figure B-3. Le but est de détecter via notre analyseur de schéma les contraintes non déclarées explicitement dans le code LDD. Pour ce faire celui-ci utilise certaines heuristiques basées sur des règles de « bonnes pratiques ». Le chapitre 4 décrit l'ensemble des heuristiques utilisées pour l'analyse du schéma. Celles-ci sont essentiellement basées sur la convention des noms.

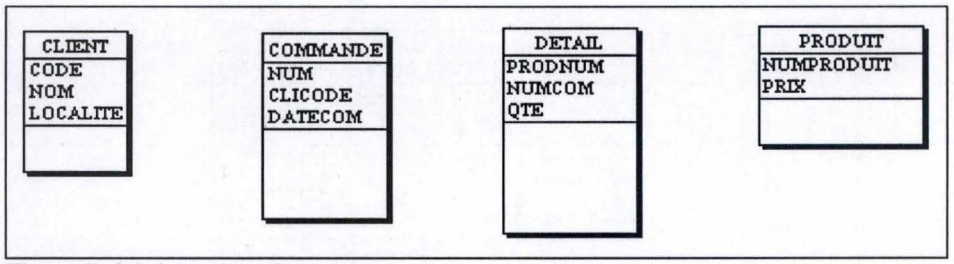


Figure B-3 Schéma à analyser.

Les heuristiques ayant été utiles pour la détection des indices sont montrées dans la figure B-4.

	Heuristique	Heuristique Vérifiée
Clé primaire	Correspondance du nom de l'attribut avec entité	
	Préfixe/Suffixe ID	
	Préfixe/Suffixe NUM	✓
	Préfixe/Suffixe CODE	✓
	Position	✓
	Type de l'attribut (INT)	
	Type de l'attribut (CHAR)	✓
	Type de l'attribut (DATE)	
Clé étrangère	Correspondance des noms	✓
	Traduction	
	Synonyme	
	Type identique	✓
	Taille identique	✓
	Taille compatible	

Figure B-4 Heuristiques utilisées par l'analyse du schéma.

ANNEXE B : Illustration des plugins

Les clés primaires et les clés étrangères potentielles ayant été identifiées par notre plugin sont présentées dans la figure B-5. Les figures B-6 et B-7 montrent les rapports ayant été produits par notre analyseur de schéma.

Clé primaire ? : CLIENT (code)
Clé primaire ? : COMMANDE (num)
Clé primaire ? : COMMANDE (clicode)
Clé primaire ? : COMMANDE (num, clicode)
Clé primaire ? : DETAIL (prodnum)
Clé primaire ? : DETAIL (numcom)
Clé primaire ? : DETAIL (prodnum, numcom)
Clé primaire ? : PRODUIT (numproduit)

Clé étrangère ? : COMMANDE (num) -> DETAIL (prodnum)
Clé étrangère ? : COMMANDE (num) -> DETAIL (numcom)
Clé étrangère ? : COMMANDE (clicode) -> CLIENT (code)
Clé étrangère ? : DETAIL (prodnum) -> COMMANDE (num)
Clé étrangère ? : DETAIL (prodnum) -> PRODUIT (numproduit)
Clé étrangère ? : DETAIL (numcom) -> COMMANDE (num)
Clé étrangère ? : PRODUIT (numproduit) -> DETAIL (prodnum)

Figure B-5 Clés primaires et étrangères potentielles identifiées par l'analyse de schéma.

DBRE - Schema Analysis - Result Primary Key for : CLIENT_COMMANDE									
N°	Attribute Name	Table Name	Corresp. with Table Name	Pos	Type	Prefix Suffix	In data ?	In pattern ?	Primary key ?
1	CODE	CLIENT	0,0%	1	CHAR	CODE	yes/no	yes/no	yes/no
2	NUM	COMMANDE	0,0%	1	CHAR	NUM	yes/no	yes/no	yes/no
3	CLICODE	COMMANDE	30,8%	2	CHAR	CODE	yes/no	yes/no	yes/no
4	PRODNUM	DETAIL	0,0%	1	CHAR	NUM	yes/no	yes/no	yes/no
5	NUMCOM	DETAIL	0,0%	2	CHAR	NUM	yes/no	yes/no	yes/no
6	NUMPRODUIT	PRODUIT	80,0%	1	CHAR	NUM	yes/no	yes/no	yes/no

Figure B-6 Rapport contenant le résultat des clés primaires potentielles obtenues à partir de l'analyse du schéma.

DBRE - Schema Analysis - Result Foreign Key for : CLIENT_COMMANDE										
N°	Attribute	Referenced Attribute	Corresp with Ref	Trans.	Syn.	Same Type	Length	In data ?	In pattern ?	Foreign key ?
1	NUM (COMMANDE)	PRODNUM (DETAIL)	50,0%	no	no	Yes	Same	yes/no	yes/no	yes/no
2	NUM (COMMANDE)	NUMCOM (DETAIL)	57,1%	no	no	Yes	Same	yes/no	yes/no	yes/no
3	CLICODE (COMMANDE)	CODE (CLIENT)	66,7%	no	no	Yes	Same	yes/no	yes/no	yes/no
4	PRODNUM (DETAIL)	NUM (COMMANDE)	50,0%	no	no	Yes	Same	yes/no	yes/no	yes/no
5	PRODNUM (DETAIL)	NUMPRODUIT (PRODUIT)	66,7%	no	no	Yes	Same	yes/no	yes/no	yes/no
6	NUMCOM (DETAIL)	NUM (COMMANDE)	57,1%	no	no	Yes	Same	yes/no	yes/no	yes/no
7	NUMPRODUIT (PRODUIT)	PRODNUM (DETAIL)	66,7%	no	no	Yes	Same	yes/no	yes/no	yes/no

Figure B-7 Rapport contenant le résultat des clés étrangères potentielles obtenues à partir de l'analyse du schéma.

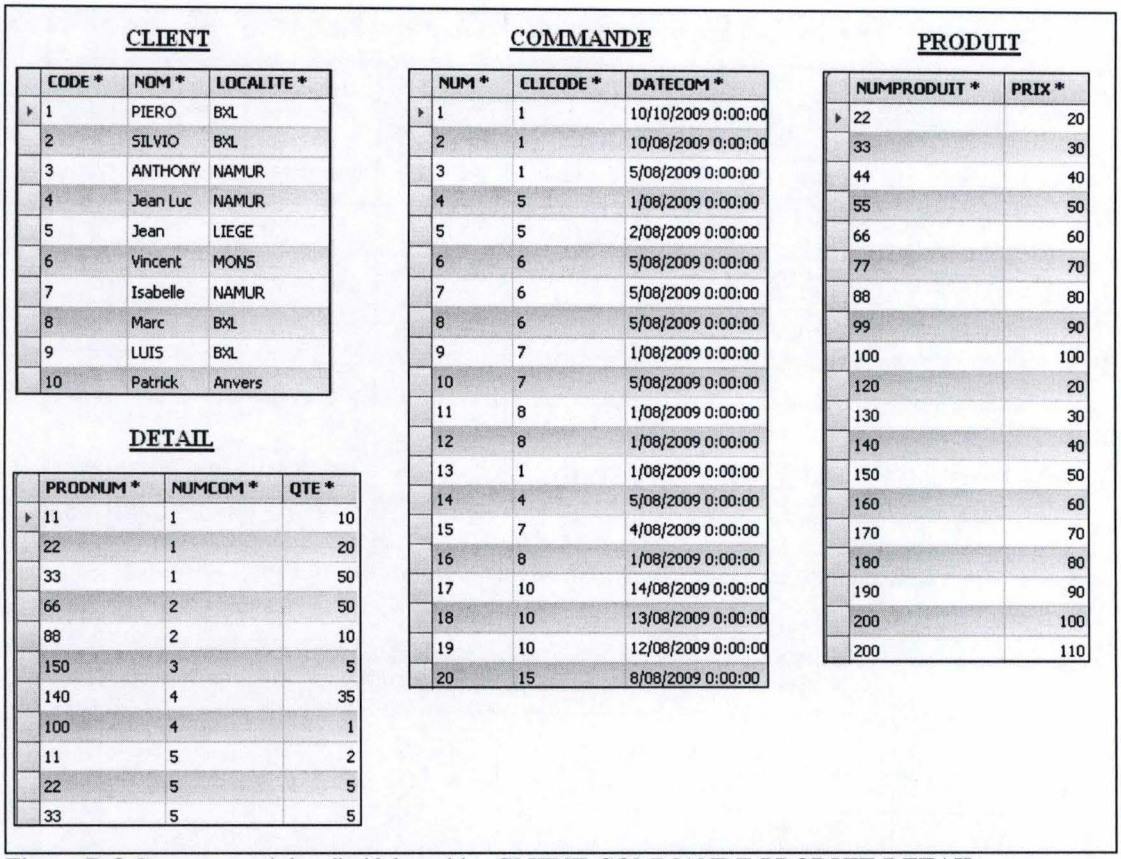
B.2.2 Analyse de données

L'ingénieur peut utiliser les données disponibles dans la base de données pour valider les hypothèses potentielles émises sur les contraintes à partir de l'analyse du schéma.
En effet, même si les données apportent une grande contribution à la validation des hypothèses. Le résultat obtenu par cette analyse peut être falsifié si les données contiennent des erreurs. Dans cette illustration nous avons utilisés un faible volume de données, celles-ci respectent en règle générale les contraintes définies implicitement.

La figure B-8 montre une partie des données contenues dans les tables. Nous y avons introduit volontairement des erreurs dans le but d'étudier l'efficacité de détection de celles-ci par nos plugins. Nous avons cependant veillé à rester en dessous du seuil de tolérance d'erreurs. Celui-ci a été fixé à 10% dans cette illustration.

C'est l'ingénieur qui se chargera de valider et de définir un pourcentage acceptable afin de valider ou de réfuter une hypothèse. Par exemple, si la contrainte est vérifiée par 85-90% des données, il est probable que cette contrainte existe, alors que si elle n'est respectée que par 10% des données, il faut la remettre en cause.

ANNEXE B : Illustration des plugins



CLIENT

CODE *	NOM *	LOCALITE *
1	PIERO	BXL
2	SILVIO	BXL
3	ANTHONY	NAMUR
4	Jean Luc	NAMUR
5	Jean	LIEGE
6	Vincent	MONS
7	Isabelle	NAMUR
8	Marc	BXL
9	LUIS	BXL
10	Patrick	Anvers

COMMANDE

NUM *	CLICODE *	DATECOM *
1	1	10/10/2009 0:00:00
2	1	10/08/2009 0:00:00
3	1	5/08/2009 0:00:00
4	5	1/08/2009 0:00:00
5	5	2/08/2009 0:00:00
6	6	5/08/2009 0:00:00
7	6	5/08/2009 0:00:00
8	6	5/08/2009 0:00:00
9	7	1/08/2009 0:00:00
10	7	5/08/2009 0:00:00
11	8	1/08/2009 0:00:00
12	8	1/08/2009 0:00:00
13	1	1/08/2009 0:00:00
14	4	5/08/2009 0:00:00
15	7	4/08/2009 0:00:00
16	8	1/08/2009 0:00:00
17	10	14/08/2009 0:00:00
18	10	13/08/2009 0:00:00
19	10	12/08/2009 0:00:00
20	15	8/08/2009 0:00:00

PRODUIT

NUMPRODUIT *	PRIX *
22	20
33	30
44	40
55	50
66	60
77	70
88	80
99	90
100	100
120	20
130	30
140	40
150	50
160	60
170	70
180	80
190	90
200	100
200	110

DETAIL

PRODNUM *	NUMCOM *	QTE *
11	1	10
22	1	20
33	1	50
66	2	50
88	2	10
150	3	5
140	4	35
100	4	1
11	5	2
22	5	5
33	5	5

Figure B-8 Contenu partiel et fictif des tables CLIENT-COMMANDE-PRODUIT-DETAIL.

La figure B-9 résume le résultat de l'analyse des données suite à l'analyse du schéma. Les rapports issus de cette analyse se trouvent dans les figures B-10 et B-11.

Clé primaire ? : CLIENT (code)
Clé primaire ? : COMMANDE (num)
~~Clé primaire ? : COMMANDE (clicode)~~
Clé primaire ? : COMMANDE (num, clicode)

~~Clé primaire ? : DETAIL (prodnum)~~
~~Clé primaire ? : DETAIL (numcom)~~
Clé primaire ? : DETAIL (prodnum, numcom)
Clé primaire ? : PRODUIT (numproduit)

~~Clé étrangère ? : COMMANDE (num) -> DETAIL (prodnum)~~
Clé étrangère ? : COMMANDE (num) -> DETAIL (numcom)
Clé étrangère ? : COMMANDE (clicode) -> CLIENT (code)
~~Clé étrangère ? : DETAIL (prodnum) -> COMMANDE (num)~~
Clé étrangère ? : DETAIL (prodnum) -> PRODUIT (numproduit)
Clé étrangère ? : DETAIL (numcom) -> COMMANDE (num)
~~Clé étrangère ? : PRODUIT (numproduit) -> DETAIL (prodnum)~~

Figure B-9 Clés primaires et étrangères potentielles identifiées par l'analyse de schéma et validées ou réfutées par l'analyse des données.

ANNEXE B : Illustration des plugins

DBRE - Data Analysis - Result Primary Key for : CLIENT_COMMANDE									
N°	Attribute Name	Table Name	Total Records	% Null	% Not Unique	Threshold %Null - %Not Unique	In data ?	In pattern ?	Primary key ?
1	CODE	CLIENT	10	0%	0%	0% - 10%	yes	yes/no	yes/no
2	NUM	COMMANDE	20	0%	0%	0% - 10%	yes	yes/no	yes/no
3	CLICODE	COMMANDE	20	0%	90%	0% - 10%	no	yes/no	yes/no
4	NUM,CLICODE	COMMANDE	20	0%	0%	0% - 10%	yes	yes/no	yes/no
5	PRODNUM	DETAIL	28	0%	71%	0% - 10%	no	yes/no	yes/no
6	NUMCOM	DETAIL	28	0%	50%	0% - 10%	no	yes/no	yes/no
7	PRODNUM,NUMCOM	DETAIL	28	0%	0%	0% - 10%	yes	yes/no	yes/no
8	NUMPRODUIT	PRODUIT	21	0%	10%	0% - 10%	yes	yes/no	yes/no

Figure B-10 Rapport contenant le résultat de l'analyse des données sur les clés primaires identifiées par l'analyse du schéma.

DBRE - Data Analysis - Result Foreign Key for : CLIENT_COMMANDE									
N°	Attribute	Referenced Attribute	Total Records	% Null	% Not Ref. Constraint	Threshold %Null - %Not Ref.	In data ?	In pattern ?	Foreign Key ?
1	NUM (COMMANDE)	PRODNUM (DETAIL)	20	0%	95%	100% - 10%	no	yes/no	yes/no
2	NUM (COMMANDE)	NUMCOM (DETAIL)	20	0%	0%	100% - 10%	yes	yes/no	yes/no
3	CLICODE (COMMANDE)	CODE (CLIENT)	20	0%	5%	100% - 10%	yes	yes/no	yes/no
4	PRODNUM (DETAIL)	NUM (COMMANDE)	28	0%	86%	100% - 10%	no	yes/no	yes/no
5	PRODNUM (DETAIL)	NUMPRODUIT (PRODUIT)	28	0%	0%	100% - 10%	yes	yes/no	yes/no
6	NUMCOM (DETAIL)	NUM (COMMANDE)	28	0%	0%	100% - 10%	yes	yes/no	yes/no
7	NUMPRODUIT (PRODUIT)	PRODNUM (DETAIL)	21	0%	29%	100% - 10%	no	yes/no	yes/no

Figure B-11 Rapport contenant le résultat de l'analyse des données sur les clés étrangères identifiées par l'analyse du schéma.

Le schéma final obtenu suite à l'analyse du schéma et des données est présenté dans la figure B-12. L'analyse de ces deux composants a permis d'éliciter les 4 clés primaires et les 3 clés étrangères implicites. En effet, la clé COMMANDE (num, clicode) n'a pas été prise en compte puisque COMMANDE (num) suffit à lui seul pour constituer un identifiant de la table COMMANDE.

De plus, les deux clés étrangères COMMANDE (num) -> DETAIL (numcom) et DETAIL (numcom) -> COMMANDE(num) faut penser à la présence d'une contrainte d'égalité (equ).

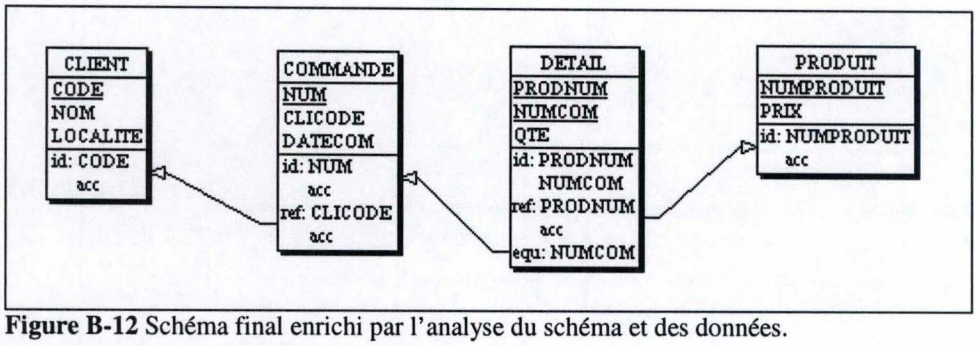


Figure B-12 Schéma final enrichi par l'analyse du schéma et des données.

B.2.3 Détection des données « erronées »

Pour la détection des éventuelles données erronées les rapports de l'analyse des données et notre générateur de requêtes apportent une aide précieuse à l'ingénieur.

En effet, les données dites suspectes (erreurs potentielles) sont les données pour lesquelles le pourcentage de violation d'une contrainte est faible et donc souvent inférieur au seuil de tolérance défini par l'ingénieur (figures B-13).

DBRE - Data Analysis - Result Primary Key for : CLIENT_COMMANDE

N°	Attribute Name	Table Name	Total Records	% Null	% Not Unique	Threshold %Null - %Not Unique	In data ?	In pattern ?	Primary key ?
1	CODE	CLIENT	10	0%	0%	0% - 10%	yes	yes/no	yes/no
2	NUM	COMMANDE	20	0%	0%	0% - 10%	yes	yes/no	yes/no
3	CLICODE	COMMANDE	20	0%	90%	0% - 10%	no	yes/no	yes/no
4	NUM,CLICODE	COMMANDE	20	0%	0%	0% - 10%	yes	yes/no	yes/no
5	PRODNUM	DETAIL	28	0%	71%	0% - 10%	no	yes/no	yes/no
6	NUMCOM	DETAIL	28	0%	50%	0% - 10%	no	yes/no	yes/no
7	PRODNUM,NUMCOM	DETAIL	28	0%	0%	0% - 10%	yes	yes/no	yes/no
8	NUMPRODUIT	PRODUIT	21	0%	10%	0% - 10%	yes	yes/no	yes/no

a) Rapport contenant le résultat de l'analyse des clés primaires par les données.

ANNEXE B : Illustration des plugins

DBRE - Data Analysis - Result Foreign Key for : CLIENT_COMMANDE									
N°	Attribute	Referenced Attribute	Total Records	% Null	% Not Ref. Constraint	Threshold %Null - %Not Ref.	In data ?	In pattern ?	Foreign Key ?
1	NUM (COMMANDE)	PRODNUM (DETAIL)	20	0%	95%	100% - 10%	no	yes/no	yes/no
2	NUM (COMMANDE)	NUMCOM (DETAIL)	20	0%	0%	100% - 10%	yes	yes/no	yes/no
3	CLICODE (COMMANDE)	CODE (CLIENT)	20	0%	5%	100% - 10%	yes	yes/no	yes/no
4	PRODNUM (DETAIL)	NUM (COMMANDE)	28	0%	86%	100% - 10%	no	yes/no	yes/no
5	PRODNUM (DETAIL)	NUMPRODUIT (PRODUIT)	28	0%	0%	100% - 10%	yes	yes/no	yes/no
6	NUMCOM (DETAIL)	NUM (COMMANDE)	28	0%	0%	100% - 10%	yes	yes/no	yes/no
7	NUMPRODUIT (PRODUIT)	PRODNUM (DETAIL)	21	0%	29%	100% - 10%	no	yes/no	yes/no

b) Rapport contenant le résultat de l'analyse des clés étrangères par les données.

Figures B-13 Rapports contenant le résultat de l'analyse des données. Ceux-ci peuvent indiquer la présence d'erreurs potentielles dans les données.

La figure B-14 montre les éventuelles données inconsistantes par rapport aux contraintes identifiées se trouvant dans les tables.

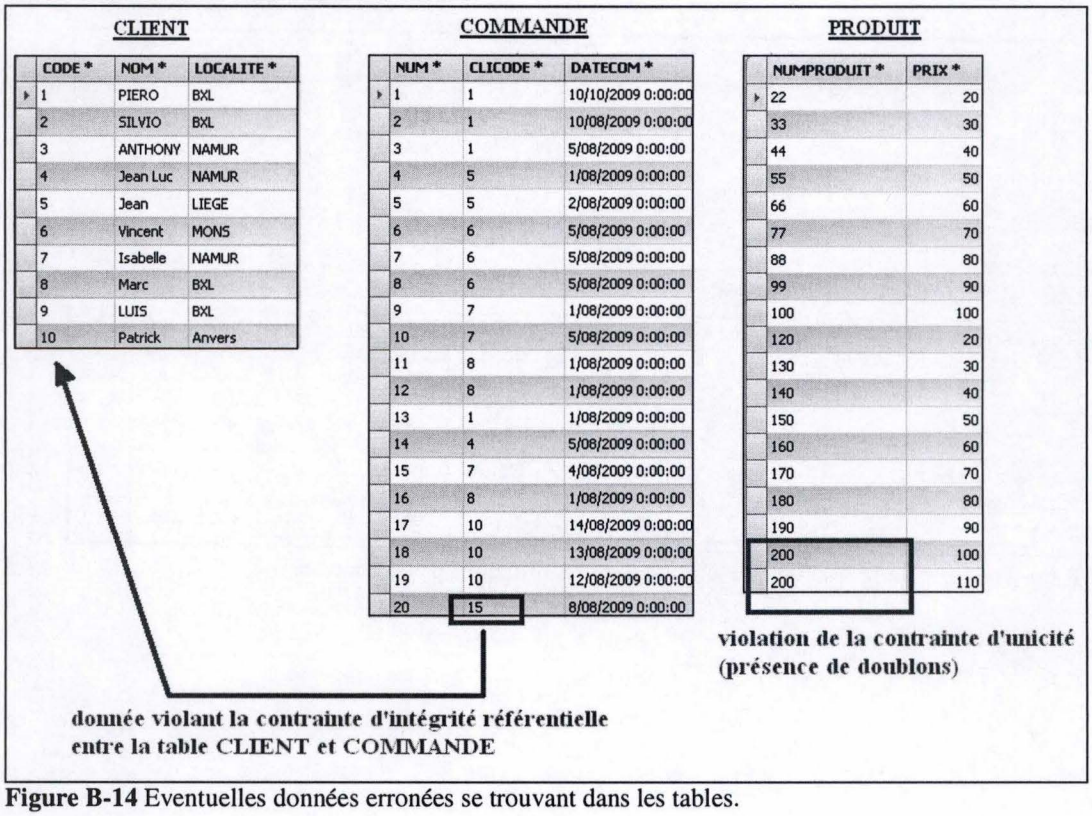


Figure B-14 Eventuelles données erronées se trouvant dans les tables.

Les requêtes générées pour la détection des données suspectes et le résultat de celles-ci se trouvent dans la figure B-15.

Rôle	Requête	Résultat
Clé primaire : PRODUIT(numproduit) Vérification des enregistrements violant la contrainte d'unicité sur l'attribut numproduit de la table PRODUIT.	`SELECT * FROM PRODUIT where NUMPRODUIT IN (select NUMPRODUIT from PRODUIT group by NUMPRODUIT having count(NUMPRODUIT) > 1);`	NUMPRODUIT * | PRIX * 200 | 100 200 | 110
Clé étrangère : COMMANDE(clicode) -> CLIENT(code) Vérification des enregistrements violant la contraintes d'intégrité référentielle entre COMMANDE(clicode) et CLIENT(code).	`SELECT * FROM COMMANDE where CLICODE not in (select CODE from CLIENT);`	NUM * | CLICODE * | DATECOM * 20 | 15 | 8/08/2009 0:00:00

Figure B-15 Requêtes générées pour la détection des données suspectes.

Rappelons que l'estimation d'une donnée erronée est un processus manuel.
En effet, malgré que la recherche des données invalides soit un processus automatique, l'évaluation du résultat obtenu et leur résolution reste du domaine de l'ingénieur.
Bien souvent, c'est au niveau de la logique business que se décidera la solution finale à apporter à ces données . Cette solution manuelle est de loin la plus précise et la plus fiable.

B.2.4 Utilité des triggers

Les triggers ont pour but de « stopper l'hémorragie ». Ce renforcement du côté de la base de données peut réduire considérablement le risque d'introduction de données incohérentes dans la base de données.

Dans cette illustration l'ingénieur à choisi d'intégrer les triggers de la figure B-17. Ceux-ci permettent d'une part de garantir la contrainte d'unicité sur la clé numproduit de la table PRODUIT et d'autre part de garantir la bonne gestion de la contrainte référentielle entre les tables COMMANDE et CLIENT.

Suite à la création de notre trigger la figure B-16 illustre la réaction du SGBD MySQL lors de l'insertion d'un enregistrement pouvant violer la contrainte d'unicité de la table PRODUIT.

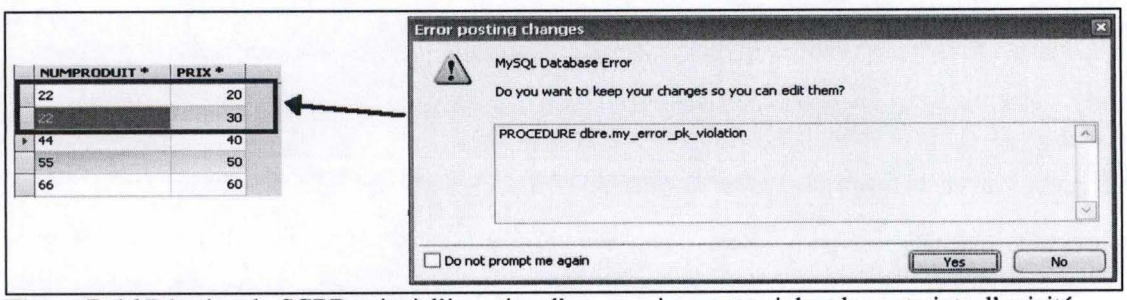
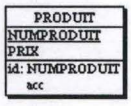


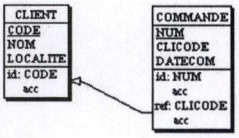
Figure B-16 Réaction du SGBD suite à l'insertion d'un enregistrement violant la contrainte d'unicité.




```
delimiter //
-- creation trigger on INSERT for Primary Key : NUMPRODUIT on PRODUIT
create trigger CHK_PK_INS_PRODUIT_7
BEFORE INSERT ON PRODUIT
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM PRODUIT WHERE NUMPRODUIT=new.NUMPRODUIT) THEN
     call my_error_pk_violation();
END IF;
END//
delimiter :

delimiter //
-- creation trigger on UPDATE for Primary Key : NUMPRODUIT on PRODUIT
create trigger CHK_PK_UPD_PRODUIT_8
BEFORE UPDATE ON PRODUIT
FOR EACH ROW
BEGIN
IF EXISTS (SELECT * FROM PRODUIT WHERE NUMPRODUIT=new.NUMPRODUIT) THEN
  IF ((old.NUMPRODUIT <> new.NUMPRODUIT)) THEN
     call my_error_pk_violation();
  END IF;
END IF;
END//
delimiter :
```

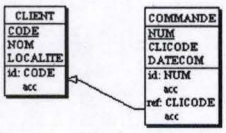
a) Triggers garantissant la contrainte d'unicité de la clé primaire numproduit de PRODUIT.




```
delimiter //
-- creation trigger on INSERT for Foreign Key (child): CLICODE of COMMANDE
create trigger CHK_FK_CHILD_IN_CLICODE_COMMANDE
BEFORE INSERT ON COMMANDE
FOR EACH ROW
BEGIN
IF NOT (new.CLICODE is null) THEN
 IF NOT EXISTS (SELECT CODE FROM CLIENT WHERE CODE= new.CLICODE) THEN
     call my_error_fk_violation();
 END IF;
END IF;
END//
delimiter ;

delimiter //
-- creation trigger on UPDATE for Foreign Key (child): CLICODE of COMMANDE
create trigger CHK_FK_CHILD_UP_CLICODE_COMMANDE
BEFORE UPDATE ON COMMANDE
FOR EACH ROW
BEGIN
IF NOT (new.CLICODE is null) THEN
 IF (old.CLICODE <> new.CLICODE) THEN
  IF NOT EXISTS (SELECT CODE FROM CLIENT WHERE CODE= new.CLICODE) THEN
     call my_error_fk_violation();
  END IF;
 END IF;
END IF;
END//
delimiter ;
```

b) Triggers assurant la gestion de l'intégrité référentielle au niveau de la table enfant (table COMMANDE).




```
delimiter //
-- creation trigger on DELETE  in parent table for Foreign Key : CLICODE of COMMANDE
create trigger CHK_FK_PARENT_DEL_CODE_CLIENT
BEFORE DELETE ON CLIENT
FOR EACH ROW
BEGIN
 IF EXISTS (SELECT CLICODE FROM COMMANDE WHERE CLICODE= old.CODE) THEN
     call my_error_fk_violation();
 END IF;
END//
delimiter ;

delimiter //
-- creation trigger on UPDATE  in parent table for Foreign Key : CLICODE of COMMANDE
create trigger CHK_FK_PARENT_UPD_CODE_CLIENT
BEFORE UPDATE ON CLIENT
FOR EACH ROW
BEGIN
IF (old.CODE <> new.CODE) THEN
  IF EXISTS (SELECT CLICODE FROM COMMANDE WHERE CLICODE= old.CODE) THEN
     call my_error_fk_violation();
  END IF;
END IF;
END//
delimiter ;
```

c) Triggers assurant la gestion de l'intégrité référentielle au niveau de la table parent (table CLIENT).

Figures B-17 Exemple de triggers générés pour garantir quelques unes des contraintes implicites de la base de données de la gestion des clients et des commandes.

ANNEXE C : Code source du programme COBOL (Client-Commande)

C.1 Première version du code source du programme COBOL/EMBEDDED SQL.

```
1:          IDENTIFICATION DIVISION.
2:           PROGRAM-ID. test.
3:
4:          DATA DIVISION.
5:           WORKING-STORAGE SECTION.
6:           01 COMMANDEDetails.
7:              02  NCOM-COM        PIC X(10).
8:              02  CLI-COM         PIC X(10).
9:              02  DATE-COM        PIC X(10).
10:
11:          01 CLIENTDetails.
12:             02 NCLI-CLIENT     PIC 9(10).
13:             02 NOM-CLIENT      PIC X(20).
14:             02 LOCA-CLIENT     PIC X(20).
15:
16:         * FLAG
17:          01 NEW-COMMANDE-FLAG      PIC X VALUE 'Y'.
18:          01 NEW-CLIENT-FLAG        PIC X VALUE 'Y'.
19:          01 UPDATE-CLIENT-FLAG     PIC X VALUE 'Y'.
20:          01 UPDATE-COM-FLAG        PIC X VALUE 'Y'.
21:          01 DELETE-COM-FLAG        PIC X VALUE 'Y'.
22:          01 DISP-LOCA-FLAG         PIC X VALUE 'Y'.
23:
24:         * VARIABLE ACCEPT
25:         * COMMANDE
26:          01 ACCEPT-NCOM        PIC 9(10).
27:          01 ACCEPT-CLI         PIC 9(10).
28:          01 ACCEPT-DATE        PIC X(10).
29:
30:         * CLIENT
31:          01 ACCEPT-NCLI        PIC 9(10).
32:          01 ACCEPT-NOM         PIC X(10).
33:          01 ACCEPT-LOCA        PIC X(10).
34:          01 ACCEPT-NCLI-OLD    PIC 9(10).
35:          01 ACCEPT-NCLI-NEW    PIC 9(10).
36:
37:          77  SW-ROW-TROUVEE                PIC  X(01).
38:              88 ROW-TROUVEE          VALUE 'Y'.
```

ANNEXE C : Code source du programme COBOL (Client-Commande)

```
39:
40:          77  SW-ROW-INSERT                    PIC  X(01).
41:              88  ROW-INSERT           VALUE 'Y'.
42:
43:          77  SW-ROW-UPDATE                    PIC  X(01).
44:              88  ROW-UPDATE           VALUE 'Y'.
45:
46:          77  SW-ROW-DELETE                    PIC  X(01).
47:              88  ROW-DELETE           VALUE 'Y'.
48:
49:          77 NCLI-MAX            PIC 9(10).
50:          77 SQLCODE-RET         PIC S9(3).
51:
52:          EXEC SQL BEGIN DECLARE SECTION END-EXEC
53:         * SQLCODE is 0 for success, 100 for no data, -1 for failure
54:          77 SQLCODE PIC S9(3).
55:
56:         * SQLSTATE is a 5 character communication code; 00xxx is success.
57:          77 SQLSTATE PIC X(5).
58:          EXEC SQL END DECLARE SECTION END-EXEC
59:
60:         * CURSOR
61:          EXEC SQL
62:               DECLARE C1-DISPLAY-LOCA CURSOR FOR
63:               SELECT B.NCOM, A.NCLI, A.NOM , A.LOCALITE
64:               FROM CLIENT A, COMMANDE B
65:               WHERE A.NCLI=B.CLI AND
66:                     A.LOCALITE =:ACCEPT-LOCA
67:          END-EXEC
68:
69:         PROCEDURE DIVISION.
70:          MAIN-PARAGRAPH.
71:         * Initial code
72:              PERFORM DO-CONNECT
73:
74:         * Main program
75:              PERFORM TRAITEMENT
76:
77:         * Use the database
78:              PERFORM DO-DISCONNECT
79:
80:         * Terminate the program
81:              GOBACK
82:              .
83:         *-----------------------------------------------------------------
84:         * The SQL connect statement must be completed with the information
85:         * appropriate to the actual JDBC driver in use.  JDBC stands for
```

ANNEXE C : Code source du programme COBOL (Client-Commande)

```
86:         * Java DataBase Connectivity, and it is the method by which PERCobol
87:         * accesses databases and database-like data sources.
88:         *
89:         * When connecting to a datasource, the jdbc:url may be
90:         * ds:data-source-name.
91:         *
92:         * jdbc:url The JDBC url to the database itself
93:         * user Usually USER      :cobol-variable-name
94:         * password Usually PASSWORD :cobol-variable-name
95:         * com.driver.name This is the classname of the driver
96:         *
97:          DO-CONNECT.
98:              EXEC SQL
99:                 CONNECT
100:                TO "jdbc:odbc:test"
101:                USER ""
102:                PASSWORD ""
103:                DRIVER "sun.jdbc.odbc.JdbcOdbcDriver"
104:             END-EXEC
105:             .
106:        *-------------------------------------------------------------
107:         TRAITEMENT.
108:        * INSERT COMMANDE
109:             PERFORM NEW-COMMANDE UNTIL NEW-COMMANDE-FLAG = "N"
110:
111:        * INSERT CLIENT
112:             PERFORM NEW-CLIENT   UNTIL NEW-CLIENT-FLAG   = "N"
113:
114:        * UPDATE NCLI CLIENT
115:             PERFORM UPDATE-NCLI-CLIENT  UNTIL UPDATE-CLIENT-FLAG = "N"
116:
117:        * UPDATE CLI COMMANDE
118:             PERFORM UPDATE-CLI-COMMANDE UNTIL UPDATE-COM-FLAG    = "N"
119:
120:        * DELETE COMMANDE
121:             PERFORM DELETE-NB-COMMANDE  UNTIL DELETE-COM-FLAG    = "N"
122:
123:        * DISPLAY COMMANDE
124:             PERFORM DISPLAY-COMMANDE-LOCALITE UNTIL DISP-LOCA-FLAG = "N"
125:             .
126:        *-------------------------------------------------------------
127:          NEW-COMMANDE.
128:             DISPLAY "CREATION NEW COMMANDE ? Y/N".
129:             ACCEPT  NEW-COMMANDE-FLAG.
130:
131:             IF NEW-COMMANDE-FLAG = "Y"
132:               DISPLAY "Enter COMMANDE details"
```

```
133:              DISPLAY "Enter - NCOM :"
134:              ACCEPT  ACCEPT-NCOM
135:
136:              DISPLAY "Enter - NCLI :"
137:              ACCEPT  ACCEPT-CLI
138:
139:              DISPLAY "Enter - DATE :"
140:              ACCEPT  ACCEPT-DATE
141:
142:         * Check CLIENT DATA
143:              PERFORM CLIENT-EXIST
144:              PERFORM RESULT-READ
145:
146:         * INSERT DATA
147:              IF ROW-TROUVEE
148:                PERFORM  COMMANDE-EXIST
149:
150:                IF NOT ROW-TROUVEE
151:                  PERFORM INSERT-COMMANDE
152:                  PERFORM RESULT-INSERT
153:
154:                  IF ROW-INSERT
155:                    PERFORM DO-COMMIT
156:                    DISPLAY "INSERT COMMANDE SUCCEED"
157:                  ELSE
158:                    DISPLAY "ERROR CREATING  COMMANDE"
159:                  END-IF
160:                ELSE
161:                  DISPLAY "ERROR INSERT COMMANDE : NCOM EXIST"
162:                END-IF
163:              ELSE
164:                DISPLAY "ERROR INSERT COMMANDE : NCLI NOT EXIST"
165:              END-IF
166:            END-IF
167:            .
168:         *-------------------------------------------------------
169:           NEW-CLIENT.
170:              DISPLAY "CREATION NEW CLIENT ? Y/N".
171:              ACCEPT  NEW-CLIENT-FLAG.
172:
173:              IF NEW-CLIENT-FLAG = "Y"
174:                DISPLAY "Enter CLIENT details"
175:
176:                DISPLAY "Enter - NOM :"
177:                ACCEPT  ACCEPT-NOM
178:
179:                DISPLAY "Enter - LOCALITE :"
180:                ACCEPT  ACCEPT-LOCA
```

ANNEXE C : Code source du programme COBOL (Client-Commande)

```
181:
182:        * get MAX NB NCLI FROM CLIENT
183:              PERFORM GET-MAX-NB-NCLI
184:              PERFORM RESULT-READ
185:
186:        * INSERT DATA
187:              IF ROW-TROUVEE
188:                ADD  1 TO NCLI-MAX
189:              ELSE
190:                MOVE 1 TO NCLI-MAX
191:              END-IF
192:
193:              PERFORM INSERT-CLIENT
194:              PERFORM RESULT-INSERT
195:
196:              IF ROW-INSERT
197:                PERFORM DO-COMMIT
198:                DISPLAY "INSERT CLIENT NB: "  NCLI-MAX " SUCCEED"
199:              ELSE
200:                DISPLAY "ERROR CREATING CLIENT"
201:              END-IF
202:              .
203:        *-------------------------------------------------------------
204:          UPDATE-NCLI-CLIENT.
205:             DISPLAY "UPDATE NCLI FROM CLIENT ? Y/N".
206:             ACCEPT  UPDATE-CLIENT-FLAG.
207:
208:             IF UPDATE-CLIENT-FLAG = "Y"
209:               DISPLAY "Enter CLIENT details"
210:
211:               DISPLAY "Enter - NCLI :"
212:               ACCEPT  ACCEPT-NCLI-OLD
213:
214:               DISPLAY "Enter - new NCLI :"
215:               ACCEPT  ACCEPT-NCLI-NEW
216:
217:               PERFORM TEST-OLD-NCLI-EXIST
218:               PERFORM RESULT-READ
219:
220:               IF ROW-TROUVEE
221:                  PERFORM TEST-NEW-NCLI-EXIST
222:                  PERFORM RESULT-READ
223:
224:             IF NOT ROW-TROUVEE
225:           * UPDATE DATA IN TABLE CLIENT
226:                  PERFORM UPDATE-NCLI-IN-CLIENT
227:                  PERFORM RESULT-UPDATE
228:
```

```
229:                IF ROW-UPDATE
230:        * UPDATE DATA IN TABLE COMMANDE
231:                  PERFORM UPDATE-CLI-IN-COMMANDE
232:                  PERFORM RESULT-UPDATE
233:
234:                  IF ROW-UPDATE
235:                    PERFORM DO-COMMIT
236:                    DISPLAY "UPDATE SUCCEED" ACCEPT-NCLI-OLD " TO "
237:                            ACCEPT-NCLI-NEW
238:                  ELSE
239:                    DISPLAY "ERROR UPDATE NCLI IN COMMANDE"
240:                  END-IF
241:                ELSE
242:                  DISPLAY "ERROR UPDATE NCLI IN CLIENT"
243:                END-IF
244:              ELSE
245:                DISPLAY "ERROR UPDATE NCLI CLIENT, NB ALREADY EXIST"
246:              END-IF
247:            ELSE
248:              DISPLAY "ERROR UPDATE NCLI CLIENT, NB NOT EXIST"
249:            END-IF
250:            .
251:        *-------------------------------------------------------------
252:         UPDATE-CLI-COMMANDE.
253:            DISPLAY "UPDATE CLI FROM COMMANDE ? Y/N".
254:            ACCEPT  UPDATE-COM-FLAG.
255:
256:            IF UPDATE-COM-FLAG = "Y"
257:              DISPLAY "Enter COMMANDE details"
258:              DISPLAY "Enter - NCOM :"
259:              ACCEPT  ACCEPT-NCOM
260:
261:              DISPLAY "Enter NEW CLI :"
262:              ACCEPT  ACCEPT-CLI
263:
264:        * Check COMMANDE DATA
265:              PERFORM COMMANDE-EXIST
266:              PERFORM RESULT-READ
267:
268:              IF ROW-TROUVEE
269:        * Check CLIENT DATA
270:                PERFORM CLIENT-EXIST
271:                PERFORM RESULT-READ
272:
273:        * UPDATE DATA
274:                IF ROW-TROUVEE
275:                  PERFORM UPDATE2-CLI-IN-COMMANDE
276:                  PERFORM RESULT-UPDATE
```

ANNEXE C : Code source du programme COBOL (Client-Commande)

```
277:
278:                 IF ROW-UPDATE
279:                   PERFORM DO-COMMIT
280:                   DISPLAY "UPDATE CLI COMMANDE SUCCEED"
281:                 ELSE
282:                   DISPLAY "ERROR UPDATING  COMMANDE"
283:                 END-IF
284:               ELSE
285:                 DISPLAY "ERROR UPDATE COMMANDE : CLI NOT EXIST"
286:               END-IF
287:             ELSE
288:                 DISPLAY "ERROR UPDATE COMMANDE : NCOM NOT EXIST"
289:             END-IF
290:           END-IF
291:           .
292:       *-------------------------------------------------------------
293:         DELETE-NB-COMMANDE .
294:           DISPLAY "DELETE COMMANDE ? Y/N".
295:           ACCEPT  DELETE-COM-FLAG.
296:
297:           IF DELETE-COM-FLAG = "Y"
298:             DISPLAY "Enter COMMANDE details"
299:             DISPLAY "Enter - NCOM :"
300:             ACCEPT  ACCEPT-NCOM
301:
302:       * Check COMMANDE DATA
303:             PERFORM COMMANDE-EXIST
304:             PERFORM RESULT-READ
305:
306:       * DELETE DATA
307:             IF ROW-TROUVEE
308:               PERFORM DELETE-COMMANDE
309:               PERFORM RESULT-DELETE
310:
311:               IF ROW-DELETE
312:                 PERFORM DO-COMMIT
313:                 DISPLAY "DELETE COMMANDE SUCCEED"
314:               ELSE
315:                 DISPLAY "ERROR DELETING  COMMANDE"
316:               END-IF
317:             ELSE
318:               DISPLAY "ERROR DELETE COMMANDE : NCOM NOT EXIST"
319:             END-IF
320:           END-IF
321:           .
322:       *-------------------------------------------------------------
323:         DISPLAY-COMMANDE-LOCALITE.
324:           DISPLAY "DISPLAY DATA COMMANDE ? Y/N".
```

```
325:             ACCEPT  DISP-LOCA-FLAG.
326:
327:             IF DISP-LOCA-FLAG = "Y"
328:               DISPLAY "Enter - LOCALITE :"
329:               ACCEPT  ACCEPT-LOCA
330:
331:           * DISPLAY DATA
332:                 EXEC SQL
333:                   OPEN C1-DISPLAY-LOCA
334:                 END-EXEC
335:
336:                 PERFORM DISPLAY-DATA-LOCA UNTIL SQLCODE = 100
337:
338:                 EXEC SQL
339:                   CLOSE C1-DISPLAY-LOCA
340:                 END-EXEC
341:             END-IF
342:             .
343:           *-------------------------------------------------------------
344:            DISPLAY-DATA-LOCA.
345:             EXEC SQL
346:               FETCH C1-DISPLAY-LOCA INTO :NCOM-COM,
347:                                          :NCLI-CLIENT ,
348:                                          :NOM-CLIENT,
349:                                          :LOCA-CLIENT
350:             END-EXEC
351:
352:             IF SQLCODE NOT = 100
353:               DISPLAY "N° COM : " NCOM-COM
354:                       "NOM CLIENT : " NOM-CLIENT
355:                       "LOCALITE   :" LOCA-CLIENT
356:             END-IF
357:             .
358:           *-------------------------------------------------------------
359:            RESULT-READ.
360:             MOVE 'N' TO SW-ROW-TROUVEE
361:
362:             IF SQLCODE  = ZEROES
363:               MOVE 'Y' TO SW-ROW-TROUVEE
364:             END-IF
365:             .
366:           *-------------------------------------------------------------
367:            RESULT-INSERT.
368:             MOVE 'N' TO SW-ROW-INSERT
369:
370:             IF SQLCODE  = ZEROES
371:               MOVE 'Y' TO SW-ROW-INSERT
372:             END-IF
```

```
373:            .
374:          *-------------------------------------------------------
375:           RESULT-UPDATE.
376:               MOVE 'N' TO SW-ROW-UPDATE
377:
378:               IF SQLCODE  = ZEROES
379:                 MOVE 'Y' TO SW-ROW-UPDATE
380:               END-IF
381:               .
382:          *-------------------------------------------------------
383:           RESULT-DELETE.
384:               MOVE 'N' TO SW-ROW-DELETE
385:
386:               IF SQLCODE  = ZEROES
387:                 MOVE 'Y' TO SW-ROW-DELETE
388:               END-IF
389:               .
390:          *-------------------------------------------------------
391:           TEST-SQLCODE-CURSOR.
392:               MOVE SQLCODE TO SQLCODE-RET
393:               .
394:          *-------------------------------------------------------
395:           INSERT-COMMANDE.
396:               EXEC SQL
397:                 INSERT INTO COMMANDE(NCOM,CLI,DATECOM)
398:                   VALUES (:ACCEPT-NCOM, :ACCEPT-CLI,:ACCEPT-DATE)
399:               END-EXEC
400:               .
401:          *-------------------------------------------------------
402:           INSERT-CLIENT.
403:               EXEC SQL
404:                 INSERT INTO CLIENT(NCLI,NOM,LOCALITE)
405:                   VALUES (:NCLI-MAX, :ACCEPT-NOM,:ACCEPT-LOCA)
406:               END-EXEC
407:               .
408:          *-------------------------------------------------------
409:           CLIENT-EXIST.
410:               EXEC SQL
411:                 SELECT NCLI
412:                 into :NCLI-CLIENT
413:                 FROM CLIENT
414:                 WHERE NCLI =:ACCEPT-CLI
415:               END-EXEC
416:               .
417:          *-------------------------------------------------------
418:           COMMANDE-EXIST.
419:               EXEC SQL
420:                 SELECT NCOM
```

```
421:              into :NCOM-COM
422:              FROM COMMANDE
423:              WHERE NCOM =:ACCEPT-NCOM
424:            END-EXEC
425:            .
426:         *-------------------------------------------------------
427:          GET-MAX-NB-NCLI.
428:            EXEC SQL
429:              SELECT MAX(NCLI)
430:              into :NCLI-MAX
431:              FROM CLIENT
432:            END-EXEC
433:            .
434:         *-------------------------------------------------------
435:          TEST-OLD-NCLI-EXIST.
436:            EXEC SQL
437:              SELECT NCLI
438:              into :NCLI-CLIENT
439:              FROM CLIENT
440:              WHERE NCLI =:ACCEPT-NCLI-OLD
441:            END-EXEC
442:            .
443:         *-------------------------------------------------------
444:          TEST-NEW-NCLI-EXIST.
445:            EXEC SQL
446:              SELECT NCLI
447:              into :NCLI-CLIENT
448:              FROM CLIENT
449:              WHERE NCLI =:ACCEPT-NCLI-NEW
450:            END-EXEC
451:            .
452:         *-------------------------------------------------------
453:          UPDATE-NCLI-IN-CLIENT.
454:            EXEC SQL
455:              UPDATE CLIENT
456:              SET NCLI=:ACCEPT-NCLI-NEW
457:              WHERE NCLI =:ACCEPT-NCLI-OLD
458:            END-EXEC
459:            .
460:         *-------------------------------------------------------
461:          UPDATE-CLI-IN-COMMANDE.
462:            EXEC SQL
463:              UPDATE COMMANDE
464:              SET CLI=:ACCEPT-NCLI-NEW
465:              WHERE CLI =:ACCEPT-NCLI-OLD
466:            END-EXEC
467:            .
468:         *-------------------------------------------------------
```

```
469:          UPDATE2-CLI-IN-COMMANDE.
470:             EXEC SQL
471:               UPDATE COMMANDE
472:               SET CLI=:ACCEPT-CLI
473:               WHERE NCOM =:ACCEPT-NCOM
474:             END-EXEC
475:             .
476:         *---------------------------------------------------------
477:          DELETE-COMMANDE.
478:             EXEC SQL
479:               DELETE FROM COMMANDE
480:               WHERE NCOM =:ACCEPT-NCOM
481:             END-EXEC
482:             .
483:         *---------------------------------------------------------
484:          DO-COMMIT.
485:             EXEC SQL
486:               COMMIT
487:             END-EXEC
488:             .
489:         *---------------------------------------------------------
490:         * Disconnect from the SQL database connection.  This allows the
491:         * JDBC driver to free any resources required for the connection.
492:          DO-DISCONNECT.
493:             EXEC SQL
494:               DISCONNECT ALL
495:             END-EXEC
496:             .
```

C.2 Seconde version du code source du programme COBOL/EMBEDDED SQL.

```
1:          IDENTIFICATION DIVISION.
2:           PROGRAM-ID. test.
3:
4:          DATA DIVISION.
5:           WORKING-STORAGE SECTION.
6:           01 COMMANDEDetails.
7:              02  NCOM-COM        PIC X(10).
8:              02  CLI-COM         PIC X(10).
9:              02  DATE-COM        PIC X(10).
10:
11:          01 CLIENTDetails.
12:             02 NCLI-CLIENT     PIC 9(10).
13:             02 NOM-CLIENT      PIC X(20).
14:             02 LOCA-CLIENT     PIC X(20).
15:
16:         * FLAG
17:          01 NEW-COMMANDE-FLAG      PIC X VALUE 'Y'.
18:          01 NEW-CLIENT-FLAG        PIC X VALUE 'Y'.
19:          01 UPDATE-CLIENT-FLAG     PIC X VALUE 'Y'.
20:          01 UPDATE-COM-FLAG        PIC X VALUE 'Y'.
21:          01 DELETE-COM-FLAG        PIC X VALUE 'Y'.
22:          01 DISP-LOCA-FLAG         PIC X VALUE 'Y'.
23:
24:         * VARIABLE ACCEPT
25:         * COMMANDE
26:          01 ACCEPT-NCOM        PIC 9(10).
27:          01 ACCEPT-CLI         PIC 9(10).
28:          01 ACCEPT-DATE        PIC X(10).
29:
30:         * CLIENT
31:          01 ACCEPT-NCLI        PIC 9(10).
32:          01 ACCEPT-NOM         PIC X(10).
33:          01 ACCEPT-LOCA        PIC X(10).
34:          01 ACCEPT-NCLI-OLD    PIC 9(10).
35:          01 ACCEPT-NCLI-NEW    PIC 9(10).
36:
37:          77  SW-ROW-TROUVEE                  PIC  X(01).
38:              88 ROW-TROUVEE          VALUE 'Y'.
39:
40:          77  SW-ROW-INSERT                   PIC  X(01).
41:              88  ROW-INSERT          VALUE 'Y'.
42:
43:          77  SW-ROW-UPDATE                   PIC  X(01).
```

ANNEXE C : Code source du programme COBOL (Client-Commande)

```
44:              88  ROW-UPDATE          VALUE 'Y'.
45:
46:          77  SW-ROW-DELETE                   PIC  X(01).
47:              88  ROW-DELETE          VALUE 'Y'.
48:
49:          77 NCLI-MAX           PIC 9(10).
50:          77 SQLCODE-RET        PIC S9(3).
51:
52:          EXEC SQL BEGIN DECLARE SECTION END-EXEC
53:         * SQLCODE is 0 for success, 100 for no data, -1 for failure
54:          77 SQLCODE PIC S9(3).
55:
56:         * SQLSTATE is a 5 character communication code; 00xxx is success.
57:          77 SQLSTATE PIC X(5).
58:          EXEC SQL END DECLARE SECTION END-EXEC
59:
60:         * CURSOR n°1
61:          EXEC SQL
62:            DECLARE C1-DISPLAY-LOCA CURSOR FOR
63:            SELECT A.NCLI, A.NOM , A.LOCALITE
64:            FROM CLIENT A
65:            WHERE A.LOCALITE =:ACCEPT-LOCA
66:          END-EXEC
67:
68:         * CURSOR n°2
69:          EXEC SQL
70:            DECLARE C2-DISPLAY-LOCA CURSOR FOR
71:            SELECT B.NCOM
72:            FROM COMMANDE B
73:            WHERE B.CLI =: NCLI-CLIENT
74:          END-EXEC
75:
76:         PROCEDURE DIVISION.
77:          MAIN-PARAGRAPH.
78:         * Initial code
79:              PERFORM DO-CONNECT
80:
81:         * Main program
82:              PERFORM TRAITEMENT
83:
84:         * Use the database
85:              PERFORM DO-DISCONNECT
86:
87:         * Terminate the program
88:              GOBACK
89:              .
90:          *-----------------------------------------------------------------
91:          * The SQL connect statement must be completed with the information
```

```
92:         * appropriate to the actual JDBC driver in use.  JDBC stands for
93:         * Java DataBase Connectivity, and it is the method by which PERCobol
94:         * accesses databases and database-like data sources.
95:         *
96:         * When connecting to a datasource, the jdbc:url may be
97:         * ds:data-source-name.
98:         *
99:         * jdbc:url The JDBC url to the database itself
100:        * user Usually USER     :cobol-variable-name
101:        * password Usually PASSWORD :cobol-variable-name
102:        * com.driver.name This is the classname of the driver
103:        *
104:         DO-CONNECT.
105:            EXEC SQL
106:              CONNECT
107:              TO "jdbc:odbc:test"
108:              USER ""
109:              PASSWORD ""
110:              DRIVER "sun.jdbc.odbc.JdbcOdbcDriver"
111:            END-EXEC
112:            .
113:        *-------------------------------------------------------------
114:         TRAITEMENT.
115:        * INSERT COMMANDE
116:            PERFORM NEW-COMMANDE UNTIL NEW-COMMANDE-FLAG = "N"
117:
118:        * INSERT CLIENT
119:            PERFORM NEW-CLIENT   UNTIL NEW-CLIENT-FLAG   = "N"
120:
121:        * UPDATE NCLI CLIENT
122:            PERFORM UPDATE-NCLI-CLIENT  UNTIL UPDATE-CLIENT-FLAG = "N"
123:
124:        * UPDATE CLI COMMANDE
125:            PERFORM UPDATE-CLI-COMMANDE UNTIL UPDATE-COM-FLAG    = "N"
126:
127:        * DELETE COMMANDE
128:            PERFORM DELETE-NB-COMMANDE  UNTIL DELETE-COM-FLAG    = "N"
129:
130:        * DISPLAY COMMANDE
131:            PERFORM DISPLAY-COMMANDE-LOCALITE UNTIL DISP-LOCA-FLAG = "N"
132:            .
133:        *-------------------------------------------------------------
134:         NEW-COMMANDE.
135:            DISPLAY "CREATION NEW COMMANDE ? Y/N".
136:            ACCEPT  NEW-COMMANDE-FLAG.
137:
138:            IF NEW-COMMANDE-FLAG = "Y"
139:              DISPLAY "Enter COMMANDE details"
```

```
140:              DISPLAY "Enter - NCOM :"
141:              ACCEPT  ACCEPT-NCOM
142:
143:              DISPLAY "Enter - NCLI :"
144:              ACCEPT  ACCEPT-CLI
145:
146:              DISPLAY "Enter - DATE :"
147:              ACCEPT  ACCEPT-DATE
148:
149:         * Check CLIENT DATA
150:              PERFORM CLIENT-EXIST
151:              PERFORM RESULT-READ
152:
153:         * INSERT DATA
154:              IF CPT > 0
155:                PERFORM COMMANDE-EXIST
156:
157:                IF CPT = 0
158:                  PERFORM INSERT-COMMANDE
159:
160:                IF CPT2 > 0
161:                     DISPLAY "INSERT COMMANDE SUCCEED"
162:                   ELSE
163:                     DISPLAY "ERROR CREATING  COMMANDE"
164:                   END-IF
165:                ELSE
166:                  DISPLAY "ERROR INSERT COMMANDE : NCOM NOT EXIST"
167:                END-IF
168:              ELSE
169:                DISPLAY "ERROR INSERT COMMANDE : NCLI NOT EXIST"
170:              END-IF
171:            END-IF
172:            .
173:         *-------------------------------------------------------
174:          NEW-CLIENT.
175:            DISPLAY "CREATION NEW CLIENT ? Y/N".
176:            ACCEPT  NEW-CLIENT-FLAG.
177:
178:            IF NEW-CLIENT-FLAG = "Y"
179:              DISPLAY "Enter CLIENT details"
180:
181:              DISPLAY "Enter - NOM :"
182:              ACCEPT  ACCEPT-NOM
183:
184:              DISPLAY "Enter - LOCALITE :"
185:              ACCEPT  ACCEPT-LOCA
186:
187:          * get MAX NB NCLI FROM CLIENT
```

```
188:              PERFORM GET-MAX-NB-NCLI
189:
190:         * INSERT DATA
191:              IF NCLI-MAX > 0
192:                ADD  1 TO NCLI-MAX
193:              ELSE
194:                MOVE 1 TO NCLI-MAX
195:              END-IF
196:
197:              PERFORM INSERT-CLIENT
198:
199:             IF CPT2 > 0
200:                DISPLAY "INSERT CLIENT NB: "  NCLI-MAX " SUCCEED"
201:              ELSE
202:                DISPLAY "ERROR CREATING CLIENT"
203:              END-IF
204:              .
205:         *-------------------------------------------------------------
206:          UPDATE-NCLI-CLIENT.
207:              DISPLAY "UPDATE NCLI FROM CLIENT ? Y/N".
208:              ACCEPT  UPDATE-CLIENT-FLAG.
209:
210:              IF UPDATE-CLIENT-FLAG = "Y"
211:                DISPLAY "Enter CLIENT details"
212:
213:                DISPLAY "Enter - NCLI :"
214:                ACCEPT  ACCEPT-NCLI-OLD
215:
216:                DISPLAY "Enter - new NCLI :"
217:                ACCEPT  ACCEPT-NCLI-NEW
218:
219:                PERFORM TEST-OLD-NCLI-EXIST
220:
221:              IF CPT > 0
222:                  PERFORM TEST-NEW-NCLI-EXIST
223:
224:              IF CPT = 0
225:         * UPDATE DATA IN TABLE CLIENT && TABLE COMMMANDE
226:                PERFORM UPDATE-NCLI-IN-CLIENT-COMMANDE
227:
228:                   IF CPT > 0
229:                     DISPLAY "UPDATE SUCCEED" ACCEPT-NCLI-OLD " TO "
230:                                              ACCEPT-NCLI-NEW
231:                   ELSE
232:                     DISPLAY "ERROR UPDATE NCLI IN COMMANDE AND CLIENT"
233:                   END-IF
234:                 ELSE
235:                   DISPLAY "ERROR UPDATE NCLI CLIENT, NB ALREADY EXIST"
```

```
236:                END-IF
237:              ELSE
238:                DISPLAY "ERROR UPDATE NCLI CLIENT, NB NOT EXIST"
239:              END-IF
240:              .
241:           *-------------------------------------------------------
242:            UPDATE-CLI-COMMANDE.
243:               DISPLAY "UPDATE CLI FROM COMMANDE ? Y/N".
244:               ACCEPT  UPDATE-COM-FLAG.
245:
246:               IF UPDATE-COM-FLAG = "Y"
247:                 DISPLAY "Enter COMMANDE details"
248:                 DISPLAY "Enter - NCOM :"
249:                 ACCEPT  ACCEPT-NCOM
250:
251:                 DISPLAY "Enter NEW CLI :"
252:                 ACCEPT  ACCEPT-CLI
253:
254:           * Check COMMANDE DATA
255:                 PERFORM COMMANDE-EXIST
256:
257:               IF CPT > 0
258:           * Check CLIENT DATA
259:                  PERFORM CLIENT-EXIST
260:
261:           * UPDATE DATA
262:                  IF CPT > 0
263:                    PERFORM UPDATE2-CLI-IN-COMMANDE
264:
265:                    IF CPT > 0
266:                      DISPLAY "UPDATE CLI COMMANDE SUCCEED"
267:                    ELSE
268:                      DISPLAY "ERROR UPDATING  COMMANDE"
269:                    END-IF
270:                  ELSE
271:                    DISPLAY "ERROR UPDATE COMMANDE : CLI NOT EXIST"
272:                  END-IF
273:                ELSE
274:                  DISPLAY "ERROR UPDATE COMMANDE : NCOM NOT EXIST"
275:                END-IF
276:              END-IF
277:              .
278:           *-------------------------------------------------------
279:            DELETE-NB-COMMANDE .
280:               DISPLAY "DELETE COMMANDE ? Y/N".
281:               ACCEPT  DELETE-COM-FLAG.
282:
283:               IF DELETE-COM-FLAG = "Y"
```

ANNEXE C : Code source du programme COBOL (Client-Commande)

```
284:              DISPLAY "Enter COMMANDE details"
285:              DISPLAY "Enter - NCOM :"
286:              ACCEPT  ACCEPT-NCOM
287:
288:           * Check COMMANDE DATA
289:              PERFORM COMMANDE-EXIST
290:
291:           * DELETE DATA
292:              IF CPT > 0
293:                PERFORM DELETE-COMMANDE
294:
295:                IF CPT = 0
296:                  DISPLAY "DELETE COMMANDE SUCCEED"
297:                ELSE
298:                  DISPLAY "ERROR DELETING  COMMANDE"
299:                END-IF
300:              ELSE
301:                DISPLAY "ERROR DELETE COMMANDE : NCOM NOT EXIST"
302:              END-IF
303:            END-IF
304:            .
305:           *-------------------------------------------------------------
306:            DISPLAY-COMMANDE-LOCALITE.
307:               DISPLAY "DISPLAY DATA COMMANDE ? Y/N".
308:               ACCEPT  DISP-LOCA-FLAG.
309:
310:               IF DISP-LOCA-FLAG = "Y"
311:                 DISPLAY "Enter - LOCALITE :"
312:                 ACCEPT  ACCEPT-LOCA
313:
314:           * DISPLAY DATA
315:           * 1ere boucle
316:              EXEC SQL
317:                OPEN C1-DISPLAY-LOCA
318:              END-EXEC
319:
320:              PERFORM DISPLAY-DATA-LOCA-1 UNTIL SQLCODE = 100
321:
322:              EXEC SQL
323:                CLOSE C1-DISPLAY-LOCA
324:              END-EXEC
325:            END-IF
326:            .
327:           *-------------------------------------------------------------
328:            DISPLAY-DATA-LOCA-1.
329:               EXEC SQL
330:                 FETCH C1-DISPLAY-LOCA INTO :NCLI-CLIENT ,
331:                                            :NOM-CLIENT,
```

```
332:                                          :LOCA-CLIENT
333:           END-EXEC
334:
335:           IF SQLCODE NOT = 100
336:         * 2eme boucle
337:             EXEC SQL
338:               OPEN C2-DISPLAY-LOCA
339:             END-EXEC
340:
341:             PERFORM DISPLAY-DATA-LOCA-2 UNTIL SQLCODE = 100
342:
343:             EXEC SQL
344:               CLOSE C2-DISPLAY-LOCA
345:             END-EXEC
346:           END-IF
347:           .
348:         *-------------------------------------------------------
349:          DISPLAY-DATA-LOCA-2.
350:           EXEC SQL
351:             FETCH C2-DISPLAY-LOCA INTO :NCOM-COM
352:           END-EXEC
353:
354:           IF SQLCODE NOT = 100
355:              DISPLAY "N° COM : " NCOM-COM
356:                      "NOM CLIENT : " NOM-CLIENT
357:                      "LOCALITE   : " LOCA-CLIENT
358:           END-IF
359:           .
360:         *-------------------------------------------------------
361:          TEST-SQLCODE-CURSOR.
362:           MOVE SQLCODE TO SQLCODE-RET
363:           .
364:         *-------------------------------------------------------
365:          INSERT-COMMANDE.
366:             EXEC SQL
367:               INSERT INTO COMMANDE(NCOM,CLI,DATECOM)
368:               VALUES (:ACCEPT-NCOM, :ACCEPT-CLI,:ACCEPT-DATE)
369:            END-EXEC
370:
371:          PERFORM DO-COMMIT
372:
373:            EXEC SQL
374:          * VERIFICATION INSERTION
375:          SELECT COUNT(*)
376:            INTO :CPT2
377:            From COMMANDE
378:                Where NCOM =:ACCEPT-NCOM  and
379:                      CLI =: ACCEPT-CLI   and
```

```
380:                    DATECOM =: ACCEPT-DATE
381:              END-EXEC
382:              .
383:           *-------------------------------------------------------
384:            INSERT-CLIENT.
385:              EXEC SQL
386:                INSERT INTO CLIENT(NCLI,NOM,LOCALITE)
387:                VALUES (:NCLI-MAX, :ACCEPT-NOM,:ACCEPT-LOCA)
388:             END-EXEC
389:
390:           PERFORM DO-COMMIT
391:
392:             EXEC SQL
393:          * VERIFICATION INSERTION
394:           SELECT COUNT(*)
395:             INTO :CPT2
396:             From CLIENT
397:                 Where NCOM =: NCLI-MAX   and
398:                       CLI =: ACCEPT-NOM  and
399:                       DATECOM =: ACCEPT-LOCA
400:              END-EXEC
401:              .
402:           *-------------------------------------------------------
403:            CLIENT-EXIST.
404:              EXEC SQL
405:                SELECT COUNT(*)
406:                into :CPT
407:                FROM CLIENT
408:                WHERE NCLI =:ACCEPT-CLI
409:             END-EXEC
410:             .
411:           *-------------------------------------------------------
412:            COMMANDE-EXIST.
413:              EXEC SQL
414:                SELECT COUNT(*)
415:                into :CPT
416:                FROM COMMANDE
417:                WHERE NCOM =:ACCEPT-NCOM
418:              END-EXEC
419:              .
420:           *-------------------------------------------------------
421:            GET-MAX-NB-NCLI.
422:              EXEC SQL
423:                SELECT MAX(NCLI)
424:                into :NCLI-MAX
425:                FROM CLIENT
426:              END-EXEC
427:              .
```

ANNEXE C : Code source du programme COBOL (Client-Commande)

```
428:        *-------------------------------------------------------
429:         TEST-OLD-NCLI-EXIST.
430:            EXEC SQL
431:              SELECT COUNT(*)
432:              into :CPT
433:              FROM CLIENT
434:              WHERE NCLI =:ACCEPT-NCLI-OLD
435:            END-EXEC
436:            .
437:        *-------------------------------------------------------
438:         TEST-NEW-NCLI-EXIST.
439:             EXEC SQL
440:               SELECT COUNT(*)
441:               into :CPT
442:               FROM CLIENT
443:               WHERE NCLI =:ACCEPT-NCLI-NEW
444:             END-EXEC
445:             .
446:        *-------------------------------------------------------
447:         UPDATE-NCLI-IN-CLIENT-COMMANDE .
448:        * UPDATE DATA IN TABLE CLIENT
449:             PERFORM UPDATE-NCLI-IN-CLIENT
450:
451:        * UPDATE DATA IN TABLE COMMANDE
452:             PERFORM UPDATE-CLI-IN-COMMANDE
453:
454:             PERFORM DO-COMMIT
455:
456:           PERFORM TEST-NEW-NCLI-EXIST
457:             .
458:        *-------------------------------------------------------
459:          UPDATE-NCLI-IN-CLIENT.
460:             EXEC SQL
461:               UPDATE CLIENT
462:               SET NCLI=:ACCEPT-NCLI-NEW
463:               WHERE NCLI =:ACCEPT-NCLI-OLD
464:             END-EXEC
465:             .
466:        *-------------------------------------------------------
467:         UPDATE-CLI-IN-COMMANDE.
468:             EXEC SQL
469:               UPDATE COMMANDE
470:               SET CLI=:ACCEPT-NCLI-NEW
471:               WHERE CLI =:ACCEPT-NCLI-OLD
472:             END-EXEC
473:             .
474:        *-------------------------------------------------------
475:         UPDATE2-CLI-IN-COMMANDE.
```

```
476:            EXEC SQL
477:                UPDATE COMMANDE
478:                SET CLI=:ACCEPT-CLI
479:                WHERE NCOM =:ACCEPT-NCOM
480:            END-EXEC
481:
482:            PERFORM DO-COMMIT
483:
484:            EXEC SQL
485:              SELECT COUNT(*)
486:              into :CPT
487:              FROM COMMANDE
488:              WHERE CLI =:ACCEPT-CLI and
489:                    NCOM =:ACCEPT-NCOM
490:            END-EXEC
491:            .
492:        *-------------------------------------------------------
493:         DELETE-COMMANDE.
494:            EXEC SQL
495:                DELETE FROM COMMANDE
496:                WHERE NCOM =:ACCEPT-NCOM
497:            END-EXEC
498:
499:            PERFORM DO-COMMIT
500:
501:            EXEC SQL
502:              SELECT COUNT(*)
503:              into :CPT
504:              FROM COMMANDE
505:              WHERE NCOM =:ACCEPT-NCOM
506:            END-EXEC
507:            .
508:        *-------------------------------------------------------
509:         DO-COMMIT.
510:            EXEC SQL
511:              COMMIT
512:            END-EXEC
513:            .
514:        *-------------------------------------------------------
515:        * Disconnect from the SQL database connection.  This allows the
516:        * JDBC driver to free any resources required for the connection.
517:         DO-DISCONNECT.
518:            EXEC SQL
519:              DISCONNECT ALL
520:            END-EXEC
521:          .  .
```